The Anatomy of a Large Scale Social Web for Internet Enabled Objects

Antonio Pintus
Centre for Advanced Studies, Research and Development in Sardinia (CRS4)
POLARIS, loc. Piscina Manna, Ed. 1
Pula (CA), Sardinia, Italy
pintux@crs4.it

Davide Carboni
Centre for Advanced Studies, Research and Development in Sardinia (CRS4)
POLARIS, loc. Piscina Manna, Ed. 1
Pula (CA), Sardinia, Italy
dcarboni@crs4.it

Andrea Piras
Centre for Advanced Studies, Research and Development in Sardinia (CRS4)
POLARIS, loc. Piscina Manna, Ed. 1
Pula (CA), Sardinia, Italy
piras@crs4.it

ABSTRACT

The ongoing evolution of the Internet of Things toward the Web of Things, where Web-enabled smart objects connect and communicate using the protocols of the Web, has raised several research issues from protocols adoption and communication models to architectural styles. In this paper we present our vision about the anatomy of a scalable architecture for a large scale social Web of Things for smart objects and the solutions adopted. Main faced issues include a reasoned exploration of design choices in conjunction with the related state-of-art analysis, technologies, concepts and social aspects behind our proposed solution. Among them, a prototype is proposed and two experimented scenarios are described. Finally, this paper reports the conclusion, challenges and future works toward the evolution of our social Web of Things architecture and tool.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services, Data sharing*; C.2.4 [**Computer-Communication Systems**]: Distributed Systems—*client-server*; H.5.3 [**Information Systems and Presentation**]: Group and Organization Interfaces—*Web-based interaction*

General Terms

Information Systems, Computer Systems Organization

Keywords

Web of Things, REST, Social Networks

1. INTRODUCTION

In recent years, the evolution of the Internet of Things (IoT) toward the, so called, Web of Things (WoT) has paved the way to new scenarios and applications where Internet-enabled objects become active actors and peers in the Web. This trend is not only matter for research papers (see, for example [3] [15] [6]), but also for real Web-enabled products now available on the market (popular examples are: Nabaztag[1], body scales and blood pressure monitors[2] or other smart objects like Chumby[3]). Although, many of them adopt proprietary (and often closed) technologies, they represent a valid expression of the current state of the commercial side of the WoT. From a different point of view, several efforts have focused their attention to the definition of architectural models and solutions toward the use and the inter-connection of Web-enabled objects using open protocols and well known architectural styles, REST and SOAP-based Web services (i.e. [1] [7] [17]).

Another aspect worth to be explored is the *sociality* applied to the WoT where also becomes relevant how to share and collaboratively use the things in the Web and how to integrate them in existing social networks like Facebook or Twitter.

This paper describes the anatomy of a Web-based, scalable architecture for a social WoT called Paraimpu. Paraimpu is a social tool with the aim to allow people to connect, use, share and compose things, services and devices in order to create personalized new applications. This work is structured as follows: the Section 2 discusses possible design choices for the system definition and, beside each topic, a reasoned analisys of the state-of-art and our proposed solutions. Moreover, related aspects of the implemented prototype are introduced. Finally, the paper presents two application scenarios followed by conclusions and future works.

2. DESIGN CHOICES

This section reports some requirements which has been considered for the system specification, including a reasoned analisys of the state-of-art and the description of the solutions and choices which have driven the design and implementation of the system architecture. Thus, some related relevant parts of the prototype are described and discussed.

2.1 Decentralized vs. Centralized

[1]http://www.nabaztag.com/
[2]http://www.withings.com/
[3]http://www.chumby.com

WoT 2011, June 2011; San Francisco, CA, USA

Despite decentralized architectures have a number of theoretical advantages like single-point-of-failure robustness and privacy/anonimity enforcing capability, if we analyze the topology of the web we discover that the degree of decentralization is quite low, the preminent topology is client-server, and even if P2P networks have gained popularity in some application areas, the idea to move computation at the boundaries of the Internet is more a niche than a main stream. Cloud computing and Software-as-a-Service demonstrate that large IT companies are concentrating computation inside large globally distributed infrastructures transforming personal computing in a mere user interface to remote computing services.

Also in the new field of Internet-enabled objects we foresee that P2P connections between objects are unlikely to occur. There a number of technical difficulties that make them unpracticable or at least unconvenient. Only a big standardization effort could achieve this but at the moment we must admit that Internet enabled objects can at most speak HTTP and provide some form of data feed available online.

We performed some experiments with decentralized architectures [18], but to get a system able to run outside the laboratory and meet real users, we decided to adopt a centralized architecture that can practically face most of the issues cited above. A centralized online service can speak multiple protocols and data formats on top of HTTP and can act as a broker to let services meet in a logical space; can be a proxy of data coming from sources and can be a relay of data to consumer services; moreover can make adaptation of data formats where required.

In a WoT tool the workload is put at the extreme. Some applications need to have samples every second, and even only one thousand sensors connected to a central server would produce 1000 * 60 * 60 * 24 = 86400000 events a day. The load is structured in many small HTTP POST messages with a keep-alive connection. Thus, one requirements for the web server is to efficiently face the C10K problem [11]. Thus, to get the system able to scale linearly we base the design on the following points:

- C10K+ capable web servers (i.e. non-blocking servers that do not map every connection to a system thread);
- database engine able to be partitioned over a grid of machines in a transparent way (sharding);
- event handling and data processing delegated to a pool of worker processes distributed among multiple processors and even different machines.

An open-source Web server and framework which satisfies our requirements is Tornado Web Server[4], a scalable, non-blocking web server and tools that powers FriendFeed[5] social network. In conjuction with the web server, we use a load balancer/reverse proxy, NGINX[6]. In this way, the system can scale, as we run multiple instances of Tornado on multiple frontend machines.

2.2 WS-* vs. REST

[4]http://www.tornadoweb.org/
[5]https://friendfeed.com/
[6]http://wiki.nginx.org/

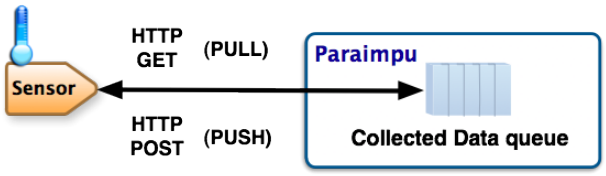

Figure 1: HTTP communication types between Paraimpu and a Sensor

Our past research experience on WoT adopting Service-oriented Architecture (SOA) with Web services (WSs) standards [18] has remarked that modeling the WoT as a network of intercorrelated processes presents some advantages: WS-BPEL, SOAP and WSDL are the standards which give the possibility to on-the-fly compose and orchestrate things. Recent [9] and past works [4] goes toward the definition of an architecture where devices are viewed as services in order to integrate a wide range of physical devices into distributed IT enterprise systems adopting a SOA. In a similar way, the projects WS4D[7] and SOCRADES [5] apply a SOA approach for embedded networks. Moreover, existing standards for WS focused on embedded devices, such as Device Profile for Web Services (DPWS)[14] confirms the general consensus and effort toward a SOA-based WoT. On the other side, some research works (i.e. [20] and [10]) use REST for IoT/WoT architectures. According to [10] experience, the programmatic complexity of WS is not well suited for the end-user to create ad-hoc applications, and also we agree with [16] that, in many cases, WS-* complexity becomes superflous and that RESTful services better support *á la Mashup* Web integrations. Thus, in our work we've chosen to mainly adopt REST as architectural style, where, in particular, things are RESTful resources modeled as follows:

- a *thing* is generally referred as a *Service*. It can be both a real thing with the meaning of a Web-enabled smart object and a "virtual thing", such as a Web resource, a social network, an external API;
- a *Service* added to the system is an univoque virtual peer of a physical or virtual thing which represents; communication between peers is HTTP-based. A Service is addressable as modeled as a RESTful resource;
- a *Service* belongs to one of the following categories: *Sensor*, which is a service that produces a data (i.e.: a thermometer or a smart body scale producing the related physical measurements, but also Foursquare[8] with its check-ins or a Pachube[9] data stream); *Actuator*, which is a Service able to receive data and perform an action (i.e.: an Arduino-based or X10 lighting control, an alarm system or Twitter, able to post received data messages, or Google Calendar for inserting agenda events).
- for a *Sensor*, data posting to the system can be performed in two ways (see Figure 1): the thing performs an HTTP POST to the system (PUSH) or, if

[7]http://www.ws4d.org/
[8]http://foursquare.com
[9]http://www.pachube.com

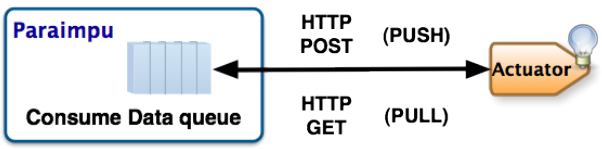

Figure 2: HTTP communication types between Paraimpu and an Actuator

the thing embeds an HTTP server, the system PULLs data through an HTTP GET to the system. For an *Actuator* can be made analogue considerations, although from an opposite point of view (see Figure 2).

In our system, RESTful resources are also *Users*, *Data*, *Connections* and all the other Paraimpu modeled entities. Each entity and data model in Paraimpu is represented by a JSON document.

2.3 Storage: Relational vs. NoSQL

Considering the scalability requirements for our system, one of the most influencing architectural aspects is represented by the data persistence system. Relational databases don't work easily in a distributed manner because joining their tables across a distributed system is difficult and they aren't designed to provide native support to data partitioning [13]. Another aspect to remark is that relational databases are schema-based and all data needs to fit into well defined tables, which does not provide the needed flexibility for an highly heterogeneous environment as the Web of Things like multimedia or data, structured or not, coming from sensors or virtual services on the Web.

In the last years a new type of database has emerged in the field of persistence, the NoSQL[10] databases family, which generally claims to provide high concurrent read-write, efficient mass data storage and access, scalability and high availability. Generally, NoSQL databases main features are: schema-free, easy replication support, simple API, not ACID, a huge data amount. Thus, it's worth taking into account that NoSQL databases don't natively support ACID transactions, they also could compromise consistency, unless manual support is provided.

Having pondered the pros and cons, in the implementation of Paraimpu we decided to adopt a NoSQL database type. In particular, we have chosen MongoDB[11] which natively stores schema-free, JSON-like documents, manages replication and fail-over failure and, among other features, it supports data arrays, dynamic queries and sharding [19]. About transactions, in Paraimpu they are contained in the context of consuming events from queues in a *readAndRemove* logic, natively implemented by database engine. Upon the MongoDB drivers, has been implemented a persistence layer (see Figure 5), in order to easily support most common used operations (like data saving and querying) wrapping native driver functions but using a simpler programming interface. MongoDB is also used both in social networks, like Foursquare and Stickybits[12] and scientific Web-based environments at CERN (see, for example, [12]).

[10]http://nosql-databases.org/
[11]http://www.mongodb.org/
[12]http://www.stickybits.com/

Table 1: Languages for data filtering

Type	Language
Numeric	Math expressions
Text	Regular expressions
JSON	Javascript
Generic XML	XPath
Atom/RSS	XPath

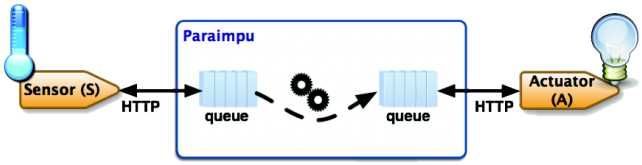

Figure 3: A connection "under-the-hood"

2.4 Connecting Things

2.4.1 Schema Enforcing vs. Data Adaptation

A big issue in connecting objects/services is how to ensure that data coming from a data source can be properly read and processed by a recipient data sink. The trivial but not practicable solution is to adopt a rigorous set of data schemas shared by all participants. In this way connections can be handled with no pains because every object consuming data is able to receive a data item and to decide what doing. Unfortunately, having a common set of data types defined and shared for all interconnected objects is far to be realistic. Objects are built from different manufacturers for vertical applications, often without the ambition to interoperate with external entities.

The only realistic assumption is to consider objects only able to produce data in one format among those commonly encapsulated in HTTP messages. So we can expect that a source can push a string containing numbers or alfanumeric values, or more structured data like JSON objects or XML instances. Pragmatically, we have divided data into the following formats: numeric, text, JSON, generic XML and Atom/RSS.

Thus we designed in Paraimpu the concepts of *filter* and *mapping*. A *filter* is a boolean expression written in a language able to process the data coming from the source (a Sensor). A data item compliant with the expression is passed to the sink (an Actuator), otherwise is ignored. If no *filter* expression is defined then all data pass.

A *mapping* is a couple of expressions in the form of *(cond,repl)* where *cond* is a boolean expression as already defined in filtering and written in a language dependant on the source (a Sensor) data type. The *repl* expression is a valid instance of the data type expected by the sink (the Actuator). As JavaScript can be used as language, a mapping for a *Numeric-to-String* connection can be written as:

```
(x>0,"Speed is "+x+" Km/h")
```

Where the varname `x` is always bound to the actual data item flowing through the connection. For the other data types format, we propose more suitable languages for data filtering, as reported in Table 1. A connection can have multiple mappings, the logic for a multiple mapping is:

```
FORALL (cond,repl) in mappings
IF cond THEN repl
ELSE CONTINUE
ENDFOR
```

Mappings and filters are defined by the user in the Connection configuration form and graphical widgets are provided to the user to compose the `repl` expression.

2.4.2 Connections Implementation

Services' data is stored by the system in dedicated *queues*, which substantially are database collections. In concrete, each *Sensor* posts produced data to an associated *Collected Data queue* (see Figure 1) and each *Actuator* consumes (gets) data from its corresponding *Consume Data queue* (see Figure 2). *Connecting* a *Sensor* to an *Actuator* corresponds to estabilish a data flow from the data queue belonging to the former to the data queue associate to the latter. This job is performed by a system process (the *Event Pumper Daemon*) which continuously copies data from Collected Data queues to Consume Data queues of connected Services, as shown in Figure 3. During the data copying process, the daemon also applies filtering and mappings to data, if required by the connection configuration. It's worth to remark that the *Event Pumper Daemon* is a process which directly accesses to the database engine, so it is possible to run several, independent, instance of it, if needed.

2.5 Social by Design

Our considerations about the social aspects in the Web of Things, can be mainly subdivided into three faces: integration with existing social networks, things sharing and, consequently, the "bookmark" vs. "discovery" philosophy.

2.5.1 Social Networks Integration

In the era of large used social networks, like Facebook or Twitter, we claim that a new social tool shouldn't be blind against them, instead, if it aims to become popular and adopted by users, it should integrate and communicate with them in some way. In Paraimpu, social networks are currently mainly used in three ways:

- *User Login/Authentication*: users can login to the system using the other social networks credentials without sharing them with Paraimpu. Thanks to OAuth[13] the user does not need to create a new account.
- *Contacts/Friends import*: the integration with other social networks also enables the import of contacts and friends from them. So, for a user, our policy is that his/her Paraimpu friends are "people that use Paraimpu" *and* "that are friends on integrated social networks".
- *Sensors/Actuators*: a social network becomes a Sensor or an Actuator (or both) just like the other things. For example Twitter can be currently used in Paraimpu as an Actuator, to post text messages, and receive data thanks to connections estabilished with Sensors.

2.5.2 Things Sharing

We think that a very relevant aspect in "socializing" the WoT is the *things sharing* between users, as remarked also in other works [8]. It plays a main role in our system and also doesn't move away too much from the idea of a *collaborative consumption* as defined in the book titled *What's mine is yours: The Rise of Collaborative Consumption* [2] where authors define it as "traditional sharing, bartering, lending, trading, renting, gifting, and swapping redefined through technology and peer communities - that is transforming business, consumerism, and the way we live." When a user adds a thing in Paraimpu he/she specifies a sharing policy for it, for example a public thing becomes available to use to all the user's friends, a private is visible only to its owner and an open thing can be used by all Paraimpu users.

[13] http://oauth.net

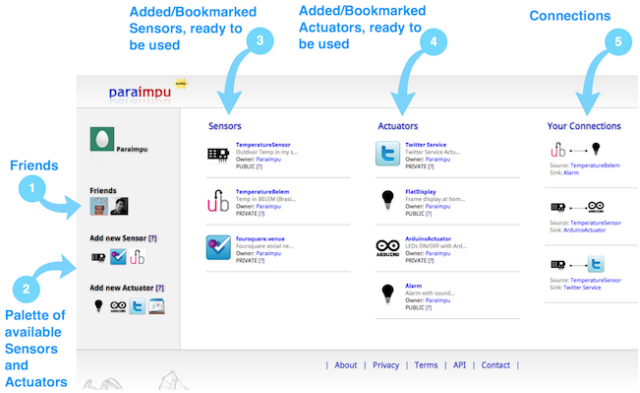

Figure 4: The Paraimpu workspace

2.5.3 Bookmarks vs. Discovery

Things sharing also impacts on another aspect: *things discovery*. For things discovery we mean the action of finding things and discovering their functionalities. We've chosen, also in this case, to furthermore simplify it. Paraimpu enable users to discover objects shared by other users/friends and to *bookmark* them. *Bookmarking* a thing adds it to the user workspace, allowing its use in a connection, for example.

3. PARAIMPU WORKSPACE

The Paraimpu user home page represents the user workspace and is structured as shown in Figure 4.

Main workspace sections are:

1. *Friends*: a list of connected friends; Selecting a friend, it is possible to discover his/her profile and his/her shared things/services. These shared things can then be bookmarked, becoming ready-to-use Services in the workspace.
2. *Sensors/Actuators palette*: The palette, split into two separate sections, shows the available Sensors/Actuators which allows the user the creation of new things in the personal workspace. Each icon represents a particular class of things (i.e. Twitter Actuator, Pachube Sensor or Arduino Actuator). Some Sensors/Actuators classes are already provided in Paraimpu such Arduino, Foursquare, Google Cal and others. Some are virtual things while other are real ones. The creation of custom Sensor/Actuator is made possible by the generic Sensor/Actuator palette's item, in particular: a *Generic Sensor* enables the creation of a generic, customizable, Sensor, a dialog opens in order to insert the

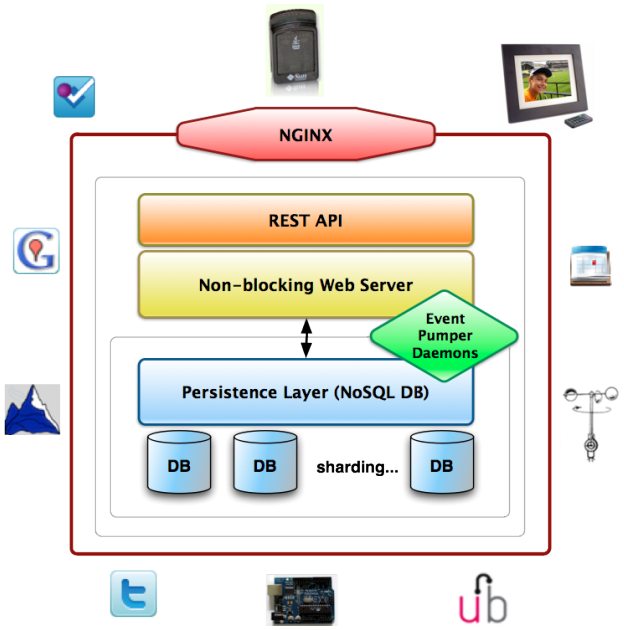

Figure 5: General architecture

Figure 6: Mappings for wind meter to Google Calendar

required info; a *Generic Actuator* enables the creation of a generic, customizable, Actuator, a dialog opens in order to insert the required info.

3. *(Added/Bookmarked) Sensors*: a list of added (using the palette) or bookmarked (from a friend profile page) Sensors in the workspace. They represent ready-to-use things. Each Sensor can be then connected to an Actuator in the workspace.

4. *(Added/Bookmarked) Actuators*: a list of added (using the palette) or bookmarked (from a friend profile page) Actuators in the workspace. They represent ready-to-use things. Each Actuator can be then used in building a connection with a Sensor in the workspace.

5. *Connections*: The list of all the created connections. A connection always involves a Sensor as the source and an Actuator as the sink. A connection can be configured and customized specifing filters and mappings (in the particular connection page).

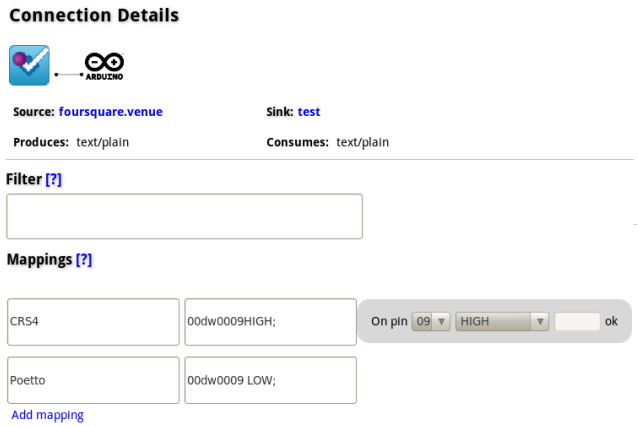

Figure 7: Mappings for Foursquare to Arduino

4. SCENARIOS

Following subsection will describe two scenarios where mixing real and virtual Internet enabled objects could help on recursive or sporadic activities.

4.1 Surfing Time

Any time, a windsurf addicted waits for good wind. Thanks to an anemometry on the preferred beach, it is possible to know the wind speed every 30 minutes. Our windsurfer registers such a sensor in Paraimpu and configures a connection with her/his Google Calendar actuator. The configuration (see Figure 6) lets to filter the minimum speed for a good wind (i.e. 30Km/h) and to schedule after a time (i.e. 2 hours) a "Windsurf" event in the agenda. With the configured Paraimpu connection, the windsurfer receives the proper reminder with the configured in advance.

4.2 Foursquare Controlled Arduino

Among the already provided type of Sensors/Actuators in Paraimpu, there are Foursquare Sensor (i.e. the last venue name checked by the user in Foursquare), and Arduino board (acting as an Actuator).

We decided to adopt a Foursquare data source instead of implementing a new mobile app with GPS capabilities, because Foursquare is a pretty popular social application.

Arduino is currently used as a class of Actuators. When a user creates a new instance of Arduino Actuator in his workspace, a sketch is generated and configured by Paraimpu for that particular instance and can be downloaded and installed in the board. Thus the generated firmware is able to receive and parse commands[14] either for setting/resetting digital outputs or for putting values to analog outputs.

In this scenario, the user connect her Foursquare checkins feed to Arduino. The logic of this connection is defined within mappings in Figure 7 and, in the example, depending on the venue the LED at pin number 9 is set to HIGH or to LOW.

5. CONCLUSIONS AND FUTURE WORK

Designing and implementing a large scale architecture for WoT imply to consider several aspects: scalability issues, data persistence, architectural styles and paradigms, things

[14]http://www.arduino.cc/playground/Code/SerialControl

abstraction and representation, connection models, social faces.

We think that mixing *sociality* and *things sharing* with WoT could really drive to a large WoT adoption and use. In this paper, we have discussed such aspects giving a state of art report along with our proposed solutions for each of the faced topics.

Finally, a prototype is presented. It lets to add, connect and share virtual and real things with users, using abstractions like Sensor and Actuators concepts. Paraimpu provides some ready-to-play ad-hoc things, like those for Foursquare, Twitter, Arduino boards. We foresee to make available new classes of things for popular devices/services like Chumby, Lego Mindstorm and so on. At the moment, about twenty active alpha users are helping us in evaluating several system issues: from usability to social aspects.

From the architectural aspect, we have designed the system mainly adopting REST style but, also thanks to the results of our past experience about WoT with SOA, we will reconsider the possibility to mix REST and WS technologies in order to use things in WS-based business processes.

6. REFERENCES

[1] O. Akribopoulos, I. Chatzigiannakis, C. Koninis, and E. Theodoridis. A web services-oriented architecture for integrating small programmable objects in the web of things. *Developments in eSystems Engineering, International Conference on*, pages 70–75, 2010.

[2] R. Botsman and R. Rogers. *What's Mine Is Yours: How Collaborative Consumption is Changing the Way We Live: The Rise of Collaborative Consumption.* Harperbusiness, 2010.

[3] K. Chung, C. Chiu, X. Xiao, and P.-Y. P. Chi. Stress outsourced: a haptic social network via crowdsourcing. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems (CHI '09)*, pages 2439–2448, New York, NY, USA, 2009. ACM.

[4] S. de Deugd, R. Carroll, K. Kelly, B. Millett, and J. Ricker. SODA: service oriented device architecture. *IEEE Pervasive Computing*, 5(3):94–96, c3, 2006.

[5] L. M. S. de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. Socrades: A web service based shop floor integration infrastructure. In *Internet of Things 2008, International Conference for Industry and Academia, Zurich, Switzerland*, pages 50–67, Mar. 2008.

[6] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier. The internet of things for ambient assisted living. In *Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations (ITNG '10)*, pages 804–809, Washington, DC, USA, 2010. IEEE Computer Society.

[7] M. Eisenhauer, P. Rosengren, and P. Antolin. A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops '09. 6th Annual IEEE Communications Society Conference on*, pages 1–3, 2009.

[8] D. Guinard, M. Fischer, and V. Trifa. Sharing using social networks in a composable web of things. In *Proc. of the First IEEE International Workshop on the Web of Things (WOT2010).*, Mannheim, Germany, Mar. 2010.

[9] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions on Services Computing*, 3(3):223–235, July 2010.

[10] D. Guinard, V. Trifa, T. Pham, and O. Liechti. Towards physical mashups in the web of things. In *Proceedings of INSS*, 2009.

[11] D. Kegel. *The C10K problem.* 2006.

[12] V. Kuznetsov, D. Evans, and S. Metson. The cms data aggregation system. *Procedia CS*, 1(1):1535–1543, 2010.

[13] N. Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, 2010.

[14] T. Nixon, A. Regnier, D. Driscoll, and A. Mensch. Devices profile for web services version 1.1. http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf, 2009.

[15] C. Outram, C. Ratti, and A. Biderman. The copenhagen wheel: An innovative electric bicycle system that harnesses the power of real-time information an crowd sourcing. In *EVER Monaco International Exhibition and Conference on Ecologic Vehicles and Renewable Energies*, 2010.

[16] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In J. Huai, R. Chen, H.-W. Hon, Y. Liu, W.-Y. Ma, A. Tomkins, and X. Zhang, editors, *WWW*, pages 805–814. ACM, 2008.

[17] A. Pintus, D. Carboni, A. Piras, and A. Giordano. Building the web of things with ws-bpel and visual tags. In *The Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010) Florence, Italy*, pages 357–360, 2010.

[18] A. Pintus, D. Carboni, A. Piras, and A. Giordano. Connecting smart things through web services orchestrations. In F. Daniel and F. Facca, editors, *Current Trends in Web Engineering*, volume 6385 of *Lecture Notes in Computer Science*, pages 431–441. Springer Berlin / Heidelberg, 2010.

[19] R. Roy. *Shard - A Database Design.* July 2008.

[20] A. S. Shirazi, C. Winkler, and A. Schmidt. Sense-sation: An extensible platform for integration of phones into the web. In F. Michahelles and J. Mitsugi, editors, *IOT*. IEEE, 2010.