Providing user support in Web-of-Things enabled Smart Spaces

Mathieu Boussard Benoit Christophe

Olivier Le Berre Vincent Toubiana

Alcatel-Lucent Bell Labs France
Route de Villejust, 91620 Nozay, France
lastname.firstname@alcatel-lucent.com

ABSTRACT

With the proliferation of connected devices, ubiquitous computing success now depends on the availability of tools and interfaces to support users in connected environments. We present a platform realizing ambient intelligence concepts that empower users in the context of Smart Spaces based on the Web of Things. This platform adopts RESTful principles to expose connected resources and defines i) a development framework to host and publish objects, ii) user interfaces and semantic based tools to find and compose real world objects iii) an application model with a set of ontologies to consume them. Our approach permits fast and simple deployments of Smart Spaces composed of Web enabled objects. With the clear objective of confronting our approach to end users, we implemented a set of user interfaces to navigate within Smart Spaces and tools to intuitively design mashups composing real world resources. We present these user support mechanisms through a semantic engine component as well as a set of browsing modalities.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces - user-centered design, ergonomics

General Terms

Design, Human Factors, Ontologies

1. INTRODUCTION

Since its introduction by Weiser in 1988, ubiquitous computing has been a fertile ground for research and technology, leading to a number of concrete advances, e.g. in mobile computing. However, the core of the vision, revolving around smart spaces in which users seamlessly consume services and information has still a long way to go. This can be explained by a number of factors, two of them being without a doubt the need for pervasive network connections and the cost of embedding advanced electronics in everyday objects. While the latter has decreased dramatically in the past few years, the former has benefited both from the Internet of Things (IoT) research field and from technology improvements, which make the Internet today a commodity available largely, and accessible to most connected devices.
It is now time to worry about providing a service layer widely deployable to foster creation and consumption of Smart Spaces, and we think that this can be embodied by the so called Web of Things (WoT). We endeavor to develop a platform and an application model that will bring this web supported vision of Smart Spaces to end-users. Our main contributions to support this vision are:

- A **framework to develop and expose on the Web location aware Smart Spaces** composed of Virtual Objects (VO).
- A **semantic driven application logic** to ease the VO composition and optimize instantiation of Smart Space applications.
- A **Web of Things browser** with multiple navigation modes designed to leverage user experiences in Smart Spaces.

Albeit frameworks to expose objects on the Web have already been designed, to the best of our knowledge no end-user oriented platform to develop and use Web enabled Smart Spaces has yet been realized.
In this paper, we depict our approach in realizing ambient intelligence concepts to support the user in the context of Web of Things enabled Smart Spaces. After discussing related works in section 2, we depict our vision as well as a user scenario in section 3 before describing in more details their realization by providing details of the framework implementation in section 4. Section 5 presents the semantic filtering tool we have developed to prevent information overload in Smart Spaces. Then it describes the browser that make use of this tool to stimulate user interactions in Web enabled Smart Spaces. Finally, section 6 provides concluding remarks and highlights our future works.

2. RELATED WORKS

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

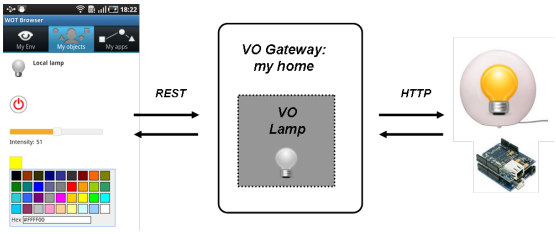


Figure 1: VO Lamp Illustration

Unlike previous realizations of the Internet of Things, the Web of Things is emerging as an open and interoperable environment to enable connected real world objects exposition and composition into applications. Most realizations of this Web of Things follow the RESTful approach [3], more adapted to resource constrained devices like sensors and actuators that are the elementary block composing the IoT. Although well targeted to enable a 'Web of Things', these realizations do not cover all Smart Spaces specificities like the requirement of offering different browsing modalities to a user, allowing accurate and practical navigation.
To cross the bridges between the existing silos, gateways enabling REST [10] functionalities on devices running proprietary protocols have been developed [3]. Pachube [1] focuses on a data-oriented vision of objects by providing web interfaces to read data produced by sensors but fails to describe and integrate them in third party mashups.
Searching real-world objects requires a description language to be used or set. Some addressed this issue by following the Service Oriented Computing (SOC) paradigm [6]. Indeed, two solutions [2, 11] used WSDL as the protocol allowing their devices to be accessed. Another interesting approach [9] uses REST architecture style as well as AtomPub protocol to describe its resources. Albeit all these languages allow to describe functionalities, their designs do not represent several aspects of real-world objects such as ownership, geographical range where the object can be accessed, access control and policies.
Like the wider Web, the WoT is de-facto accessible through a Web browser [3]. However, such an interface is not adapted to the consumption and composition of objects by nomadic users in Smart Spaces. Authors of [5] designed an interface for portable devices which supports object to object connection and composition through predefined templates that can be reused to achieve a particular goal (e.g: setting up a public presentation). However, without filtering, this interface fails to scale and composition of objects in 'object crowded' environments becomes complex.

3. VISION & MOTIVATING SCENARIO

In our work, we have focused on the user experience while interacting with WoT enabled Smart Spaces. This basically induced two intertwined aspects: enabling smart spaces that rely on WoT technological approach, and providing mechanisms and interfaces to augment the user experience in these spaces.

3.1 Overview

The realization of WoT-based Smart Spaces relies on the concept of *Virtual Objects* (VOs). In our work, a VO is an artifact representing the connected device, both to users through a digital representation, and to other software components (other VOs, applications, etc) through a processable description and an exposed HTTP API (see Fig.1). For example, a connected lamp is represented through a Lamp VO exposing:

- ***a user representation*** for visualization and interaction (described in HTML),
- ***REST APIs*** to enable the use of exposed functionalities and retrieve the HTML representation,
- ***an OWL-based semantic description*** to support reasoning on the composition or mash-up of this lamp with other VOs or services.

User support is embodied by a user interface which aims at providing a single control point for interacting with resources of a smart environment, as well as enabling composition of these resources to create ambient intelligence. Akin to the approach in [5], we opted for a browser-like interface, which presents the advantage of enabling 'recombinant computing' by being agnostic to the resources to be served. Using Web-based technology also eases the adoption by application developers, and allows for a broad availability of such applications that need to be instantiated over several connected resources.
In our example, the lamp VO includes an HTML based UI that is displayed in the interface to compose it with other objects or consume it in *applications* that can be instantiated from templates (see Fig.1). In this context, it is important to support the user by filtering/pre-selecting most relevant VOs from the list of available resources to complete these compositions. Typically, the user could select an application that provides visual notification in the environment upon a certain event (e.g. an incoming phone call), the interface would allow retrieving and configuring the application by selecting relevant VOs, in our case a phone and a lamp.

3.2 Use Case

Even if our platform is agnostic to the type of smart space it exposes (Smart Home, Smart Office, Smart City) the applications we developed and the objects we prototyped correspond to a Smart Home environment. In our illustrative use case Ben, the user, quickly ***discovers and interacts*** with objects (lamps, TVs, webcams,...) in his environment though a browser (either running on a Desktop or on a Smartphone). He then, through the same browser, ***composes his TV with a webcam*** to watch the video recorded by the webcam on its local television. The browser only proposes 'compatible' objects to compose, filtering out incoherent compositions.
Then, Ben configures the ***'Phony' application*** that issues visual signals in the environment upon an incoming phone call.
In the rest of this section we detail the different processes that are running in background, invisible to the user, to realize this use case. We refer to Fig. 2 to detail this use case and explain how and by which components, user interactions are handled.
Discovery and Use: As Ben enters a WoT-enabled Smart Space his WoT browser requests VO gateways resolution (Fig.2 [1]), based on his location and profile. The Resolver returns URLs of the relevant VOs gateways (Fig.2 [2]), from

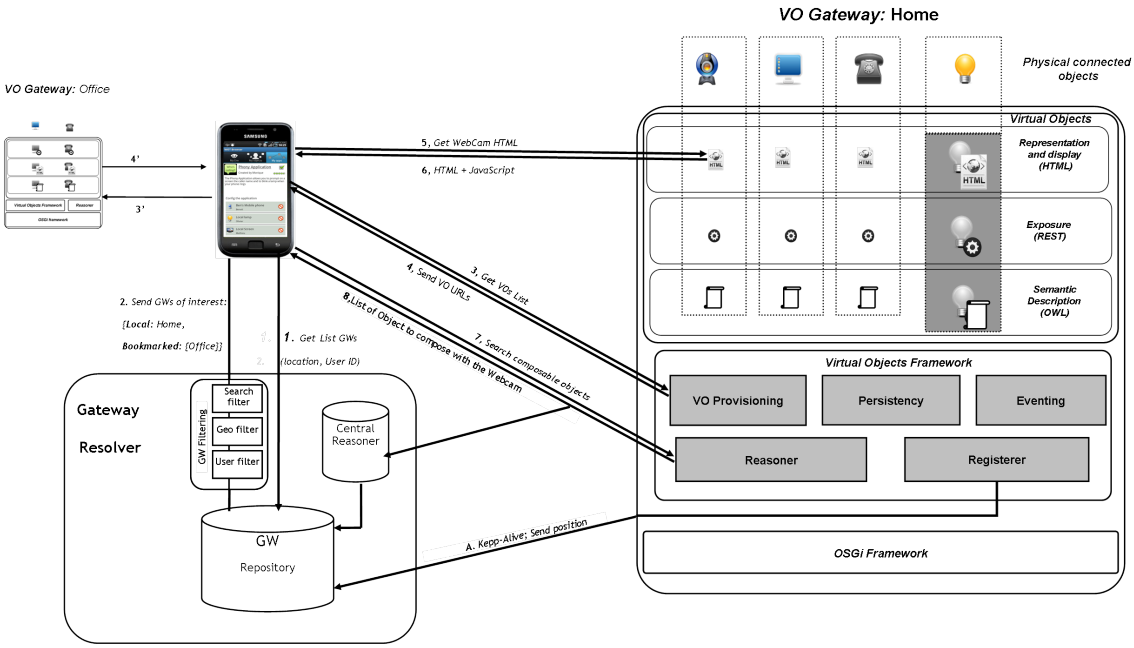


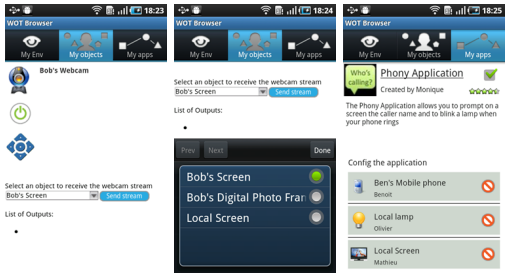
Figure 2: Web Of Things Platform

which the browser retrieves the list of local and bookmarked VOs (Fig.2 [3,4]). The latter are represented in the user interface. If Ben selects the local lamp, the HTML description is retrieved from the corresponding VO and rendered (see Fig. 1). Interacting with controls in the HTML view enables to pilot – via the REST APIs – the physical object (e.g. turn on/off).

Composition: Now at home, Ben wants to use a distant webcam to redirect its video feed on the local WoT-enabled TV. Using the browser, he retrieves the HTML representation for the Webcam VO (Fig. 3-a) which includes javascript-enabled controls (Fig.2 [5,6]). When selected, such controls trigger a callback on the browser (Fig.2 [7]) , which retrieves the semantic description associated with the functionality of the object and performs a semantic-based filtering of compatible objects in the user environment and bookmarks (Fig.2 [8]) before requesting user selection(Fig. 3-b)

Application Configuration: Finally, Ben configures the 'Phony' application via the configuration page depicted by Fig.3-c., and selects which VOs will be used in the composition. This enablement is realized by the semantic matching algorithm, that tries to find the available objects delivering one of the features expected by the application – in our example the application requires specific types of VOs, namely a phone, a lamp and a screen. Ben selects his phone, the local lamp and the local screen to configure the application. Upon reception of an 'incoming call' event (handled by the 'Eventing' component of the VO Framework) from the phone VO, the application logic triggers the lamp VO (it makes it blink with a dedicated color code depending on the caller-id) and displays a notification on the screen.

Each of the framework components involved in the discovery and utilization of VOs are detailed in section 4. Section 5 presents the semantic filtering built on top of this framework



(a) (b) (c)

Figure 3: WoT Browser

to help user composing objects and configuring applications.

4. ENABLING WOT-BASED SMART SPACES

We designed a software framework to enable WoT-based smart spaces, revolving around the embodiment of the Virtual Object concept. This includes designing the VOs themselves, providing a development framework and a hosting component (VO gateway) that represents the smart space, as well as a centralized component that enables the resolution of which gateway to connect to depending on the user profile (location, bookmarked spaces,. . .).

4.1 Virtual Object Framework

The VO Framework is built on top of Equinox, an implementation of OSGi Framework R4.2 specifications [4], and is therefore composed of OSGi bundles. We follow this approach because it provides an easy solution to enrich the set of functionalities and permits the implementation of VOs by third parties. The VO Gateway is the resulting software run-

time that includes the VO Framework as well as VO implementations (i.e: HTML file, HTTP APIs and OWL description) of objects in the considered Smart Space (e.g: lamp, phone, TV Screen, Mail Box). Furthermore, OSGi mechanisms allow dynamic instantiation of new VO by adding required OSGi bundles at runtime.

VO Provisioning: The VO provisioning component manages the life-cycle of VOs and assigns them URLs. Since each VO must implement an OSGi service called *VirtualObject*, it is possible to catch a VO instantiation or deletion event using OSGi service discovery mechanism, and thus keep up to date the list of VOs handled by a VO Gateway. The VO Provisioning element also exposes APIs that list instantiated VOs, instantiate new ones and access to their attribute values.

Persistency: The persistency component is used to automatically instantiate VOs when the gateway is starting. VO states are persisted into files so that they can be restored.

Gateway Registerer: When the Gateway starts, the Registerer informs the Gateway Resolver (detailed in section 4.3) that the Gateway is now available. It also indicates its current location. Afterwards, the Registerer sends periodical keep-alive requests.

Eventing: The eventing component notifies clients of any changes on a VO. It defines the *NotificationEngineService* OSGi service which contains two features that post VO notifications and retrieve the event channel associated to a given VO. Each VO is responsible to disseminate its events channel through its 'eventchannel' attribute. The current implementation relies on *Comet long poll* mechanism [6], meaning that any client interested in changes on a dedicated VO should perform an HTTP GET on its event channel until a change occurs and resend it to catch the next one.

Reasoner: This tool filters objects based on their semantic (see section 5 for more details). It has been implemented based on Pellet DL-reasoner [8] and offers a standard HTTP API that returns a list of matching objects (e.g. a webcam) following some requirements (e.g. ability to output a video stream). This engine is coupled with a centralized reasoner hosted in the Gateway Resolver that allows to get semantic information from remote smart spaces (e.g. a bookmarked webcam residing in another smart space).

4.2 Implementing a Virtual Object

Thanks to the VO Framework, the implementation of a VO is almost immediate: a dedicated OSGi bundle is defined for each type of VO and can be loaded dynamically at runtime on a gateway. To support several instantiations of the same kind of VO, the OSGi bundle should be declared as a Factory Component [4] and implement the *VirtualObject service* defined by the framework. This bundle is also responsible for the actual communication and synchronization with the Real World Object it represents.

For instance, we implemented the Lamp interface in Arduino [1], which can embed a small HTTP server. Its VO implementation is thus capable to receive commands from the end-user through its REST API and transmit them to the real lamp communicating with an HTTP server embedded on the Arduino electronic board (changing the lamp color for example). A private HTTP API allows a dual mode by notifying the VO with the state of the real lamp (notifying the VO that the real lamp has been physically switched on/off). Finally, the VO implementation is also responsible to serve its HTML representation (Fig. 1) and OWL description.

[1] arduino, `http://www.arduino.cc`

4.3 Gateway Resolver

The Gateway Resolver keeps up to date the list of running VOs Gateways, and makes them accessible to end-users. A filtering mechanism is implemented to only return a list of relevant Gateways considering user location, profile and requests. It also has been implemented on top of Equinox Framework and supports the following components:

Gateway Repository: The Gateway repository component maintains the list of running VO Gateways. It provides a RESTful HTTP API allowing gateways to register/unregister themselves at startup/shutdown and clients to request gateways they are interested in by using the filtering component described below. If a VO Gateway did not send a keep-alive message recently, the Gateways Repository assumes that the VO Gateway went offline improperly and removes it from its list.

Filtering: This component filters the list of VO Gateways returned to the user by discarding those matching none of the conditions expressed in the parameters of the HTTP request. The final list is then composed of gateways matching at least one of these conditions. As an instance, the geo-location filter selects the list of gateways accessible in a given area defined by latitude, longitude and radius parameters. This implies that to be selected, a gateway has to define its location and that this location has to be in the area defined in the request. We implemented two additional filters to permit the selection of bookmarked gateways and a gateway search engine.

5. MECHANISMS AND INTERFACES FOR USER SUPPORT

If the enablement of smart spaces is a first step, the second one consists in providing the means for an end-user to navigate through such spaces. By navigation means, we intend here the tools and interfaces allowing objects to i) be discovered and filtered for a dedicated use, ii) be combined together or associated within web applications and iii) obviously be represented (i.e. shown) and accessed.

All these steps require that objects from the real-world be precisely described in order to know what they offer (i.e. functionalities), for whom (e.g. is the user allowed to see and interact with an object), in which geographical area (e.g. is the object available in a dedicated smart space or not) and which representation modality to use.

5.1 Connected objects description model

The scope of potential real-world objects is so large that a standard to provide definitions for every object would be impractical. Therefore, using real-world objects in web applications requires a description language. In some ways, we may decide to re-apply service description languages (e.g. WSDL, WADL) to real-world objects. Indeed, these languages – designed to describe functionalities – could be used to represent what the real-world objects provide. However, limitations inherent to these language designs would not (or insufficiently) allow representation of real-world object specificities such as ownership, geographical range where the

object can be accessed, access control and policies.
Describing real-world objects must then be done through a dedicated model capturing their semantics. While we could have provided extensions to the SOC-based languages (e.g. a new set of WS-* dedicated for real-world objects), we have preferred to define a description language based on models allowing the use of first order logic, especially in order to allow additional knowledge to be obtained by deductive reasoning on given facts (i.e. inference). Therefore, we have developed a set of ontologies covering particular facets of an object, and providing the necessary properties to interlink them. The *vo-functional* ontology defines the generic classes and rules (using OWL[2] and SWRL[3]) allowing an object to describe its internal finite state machine. As an example, we defined here the 'Functionality' concept as an entity taking or reacting to 'inputs'; generating 'outputs'; available in 'state' and potentially changing the overall state of an object.
The *vo-structures* ontology is a collaborative repository shared amongst different object providers. It allows to define or find ontological structures and to affect them to 'inputs' or 'outputs' concepts mentioned previously. By making this ontology accessible to anyone, we ensure a common repository where, in the end, all connected objects interoperate.
The third developed ontology, called *vo-location*, defines geolocation concepts and allows objects to geographically belong to a smart place. Finally, the *vo-core* ontology links all the others as well as some that we have reused (e.g. FOAF[4]) through ontological properties (e.g. the object 'A' is in the place 'P', where 'A' is defined in 'vo-functional' and 'P' is defined in 'vo-location').

5.2 Supporting the user through semantic reasoning algorithms

Using OWL to build our models leads to the ability for semantic engines like [8, 7] to logically process the knowledge contained in each real-world object description. It subsequently permits the creation of Ambient Intelligence tools to support users in smart spaces. We have developed one tool in order to facilitate object composition by suggesting only composable objects matching some requirements. As an example of use, this tool can list all the screens a user has access to when searching for devices to display a video stream.
This semantic filter suggests object composition by analyzing how outputs produced by an object (e.g: a webcam) correspond to the inputs consumed by other objects (screens, TVs, lamps,...). In our approach, both inputs and outputs are modeled through the *vo-structure* ontology. Therefore, when the user decides to compose a webcam with another object, the filtering tool first retrieves the structure corresponding to the webcam outputs (further denoted *expected structure*). Then it analyzes the descriptions of objects available in accessible smart spaces, to find those accepting – as one of their functionalities inputs – outputs of this *expected structure*. Each time similarities are found between the *expected structure* and one object's input, a similarity score is computed. Finally, the algorithm returns a list of matching objects ranked by similarity score. Obviously, this algorithm is coupled with other filters as defined in section 4.1, in order to compare ontological structures used or produced by objects a user has access to. Notably the algorithm uses location and policy to pre-filter a list of meaningful objects.

[2]Ontology Web Language, `http://www.w3.org/TR/owl-ref/`
[3]Semantic Web Rule Language, `http://www.w3.org/Submission/SWRL/`
[4]FOAF Vocabulary Specification, `http://xmlns.com/foaf/0.1`

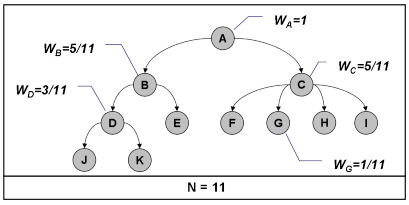


Figure 4: Weighted structure

Ontological structure preprocessing: To be able to compare ontological structures and to give a similarity score, some preprocessing steps are first required.
Ontological structures are graphs where nodes are ontological concepts and edges are ontological properties. The first step is the transformation of ontological structure from graphs into trees to avoid cycles that could lead to a non deterministic algorithm when analyzing a *vo-structure* (e.g. entering into an infinite loop when analyzing a cycle).
Once the *vo-structure* is shaped as a tree, the second step is then to weight each tree node (i.e. concept) to further establish the similarity score. The formula used to assign a weight to a node n reflects the total number of nodes of the tree as well as all the descendants of n:

$$W_n = \frac{1+D(n)}{N} \text{ with } \begin{cases} N = \textit{ total number of nodes} \\ D(n) = \textit{ all descendants of } n \\ D(root) = N - 1 \end{cases}$$

Fig. 4 shows the example of a structure transformed into a tree and how node weights are assigned.
Ranking algorithm: Recall that our algorithm aims at scoring object compositions. Then, once these preprocessing steps are done, the algorithm compares the tree generated by the *expected structure* with trees obtained from other objects. Comparison consists then of discovering if the whole tree (complete match) or if some sub-trees (partial match) of the *expected structure* can be found in object compositions' trees. Each time sub-trees are found the algorithm associates a matching score.
We compare weighted trees by transforming them into arrays. Arrays are filled by tree nodes, in decreasing weight order. We then search correspondences between the reference array representing the expected structure and the arrays corresponding to available objects.
The algorithm terminates when it has walked through all the arrays or when all the structures of the reference array have been matched. It gives a list of matching objects with an indication on how well they match the expected structure.

5.3 A WoT browser interface

Using Web standards allowed us to use a generic web browser to quickly test our platform. Even today, compliance with legacy browsers ensures that a large user base is able to navigate the WoT without having to launch a specific application. Nevertheless, when '*in-situation*', the user should be provided with a more context-aware interface to

interact with objects of his environment.
Therefore we designed a WoT browser for Android based mobile devices. This smartphone application proposes two navigation modes and provides to the user a simple switch between them:

- the overview mode displays every object attached to the visited gateway, provides a bookmark manager tab and allows users to compose objects and to instantiate applications that orchestrate several objects (see Fig 3-c),

- the augmented reality mode helps the user to discover and bookmark VOs corresponding to real world objects in his immediate environment.

Overview mode: The overview mode lists objects that are hosted by the closest gateway as well as those already bookmarked belonging to remote environments. This navigation mode has been developed as it was adapted to nomadic users, frequently interacting with distant objects (for instance: monitoring the state of home appliances offered through the Web). It can also be used to configure object composition and application using different objects. When it renders a VO description (Fig. 3-a), the browser runs the script included in the served HTML file to intercept external API calls and redirect them to the Reasoner. This javascripted hook is used to filter out the objects that do not provide required features allowing only relevant objects to be suggested for composition. Therefore, the interface displays to Ben only relevant (compatible) local objects to combine with his bookmarked home webcam (Fig. 3-b).

Augmented Reality mode: Some Smart Spaces may be composed of so many devices that presenting all of them to an end-user may result in information overloading. Using an augmented reality representation allows to significantly reduce this risk, by restricting the set of browsed elements to those that are captured by the smartphone camera. Users could accurately select a VO by simply pointing the camera at the corresponding real world object - e.g. when an application requires an object of type lamp, one could intuitively indicate his preferred lamp by pointing his camera towards it. This view is therefore the most adapted in environment where a plethora of objects provide the same functionality. The augmented reality mode first identifies the object captured by the camera and downloads the associated VO description from the Gateway. Two solutions are implemented to support object identification:

- An object recognition module integrated with the local VO Gateway identifies objects using pattern comparison algorithm (patterns are recorded in a local database hosted on the gateway). Albeit being simple to use, this solution quickly showed its limits, in particular with objects having similar shapes.

- Object tagging, e.g. using markers such as QR-codes, is preferred in object crowded environments to limit the risks of mismatch.

In order to validate our browser and the tools we developed, we will soon carry out usability evaluations and present these interfaces to end-users to collect their feedbacks.

6. CONCLUSION

This paper presented early results on using a Web of Things approach to realize Smart Spaces, and the accompanying mechanisms to support user experience in the resulting rich environments. Our approach relies on the concept of Virtual Object, the accompanying mechanisms and the models exposing semantics of the underlying connected objects. We prototyped user support mechanisms under the form of a semantic reasoning component that helps filtering out most irrelevant objects for a given composition, and a set of user interfaces to discover, interact with and compose objects in rich applications. The different components have been prototyped and are being used to evaluate the user experience and concepts when accessing and interacting with applications involving objects of the real world. Furthermore, the modular structure of the overall framework allows pursuing the work on related topics, in particular privacy and trust considerations when navigating in Smart Spaces.

7. REFERENCES

[1] Pachube. http://www.pachube.com.

[2] L. de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. Socrades: A web service based shop floor integration infrastructure. In *The Internet of Things*, volume 4952 of *Lecture Notes in Computer Science*, pages 50–67. Springer Berlin / Heidelberg, 2008.

[3] D. Guinard. Towards the web of things: Web mashups for embedded devices. In *In MEM 2009 in Proceedings of WWW 2009. ACM*, 2009.

[4] P. V. Jeff McAffer and S. Archer. *OSGi and Equinox: Creating Highly Modular Java Systems*. 2010.

[5] M. W. Newman, J. Z. Sedivy, C. M. Neuwirth, W. K. Edwards, J. I. Hong, S. Izadi, K. Marcelo, and T. F. Smith. Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*, DIS '02, pages 147–156, New York, NY, USA, 2002. ACM.

[6] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing. *Communications of the ACM*, 46:25–28, 2003.

[7] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '87, pages 237–249, New York, NY, USA, 1987. ACM.

[8] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5:51–53, June 2007.

[9] V. Stirbu. Towards a restful plug and play experience in the web of things. In *Proceedings of the 2008 IEEE International Conference on Semantic Computing*, pages 512–517, Washington, DC, USA, 2008. IEEE Computer Society.

[10] E. Wilde. Putting things to rest, 2007.

[11] E. Zeeb, A. Bobek, H. Bohn, S. Prueter, A. Pohl, H. Krumm, I. Lück, F. Golatowski, and D. Timmermann. Ws4d: Soa-toolkits making embedded systems ready for web services. Limerick, Ireland, 2007.