On the complementarity of Triple Spaces and the Web of Things

Aitor Gómez-Goiri
Deusto Institute of Technology - DeustoTech
University of Deusto
Avda. Universidades 24, 48007
Bilbao, España
aitor.gomez@deusto.es

Diego López-de-Ipiña
Deusto Institute of Technology - DeustoTech
University of Deusto
Avda. Universidades 24, 48007
Bilbao, España
dipina@deusto.es

ABSTRACT

The Internet of Things (IoT) enables communication among real-world things and devices through Internet. So far, IoT research has focused on allowing such communication through different protocols and architectures. Some of these architectural approaches are Web of Things (WoT) and Triple Space (TS) which are both resource oriented architectures. This work analyses and compares both approaches and outlines the scenarios in which they will be more useful. Particularly, it outlines how some of the drawbacks of WoT in the discovery and cooperation aspects may be complemented by integrating with TS.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; D.2.11 [**Software Architectures**]: Patterns—*blackboard*

Keywords

triple space, web of things, resource oriented architecture

1. INTRODUCTION

In recent years, Internet of Things (IoT) is becoming a reality due to the increasing number of everyday objects containing embedded devices. While in the beginning more effort was put on those devices' connectivity issues, materialized in the spread of technologies such as Zigbee, 6LoWPAN or Bluetooth, nowadays the community is focusing on higher levels of the architecture. These architectures define models in which the applications are built using the capabilities of these objects. Two of these models are Web of Things and Triple Space.

On the one hand, Web of Things (WoT) is an approach which fully integrates real-world objects and devices in the web by embedding web servers into them and adopting the REST architectural style to provide information on demand. This style is highly decoupled because it focuses on accessing in a simple way to the data itself (i.e. HTTP verbs and XML/JSON as output) and not in complex protocols of communication between objects, making the services reusable and easily understandable by developers.

On the other hand, Triple Space Computing (TS) is a paradigm where Semantic Web techniques are used to define the knowledge which is exchanged using a distributed shared space. The idea of sharing information through a common space corresponds to the *blackboard model*, used in context aware environments, which makes each process very autonomous from the rest. The Semantic Web defines the information in a very expressive and machine understandable way (new data can be even inferred), and it has also been widely adopted in context-aware environments. Thus, TS can be seen as the conjunction of two well accepted data distribution and modelling standards.

Both WoT and TS can be considered resource oriented solutions since they put emphasis in giving access to data resources. However, their differences make also them suitable for different purposes. While TS enables expressive queries, dynamic discovery and non human-mediated cooperation among objects, WoT adopts the scalable properties of the World Wide Web and it is entirely based on web standards. In this paper we explain how to take advantage of both approaches, bringing together the best of both worlds, and illustrate the benefits of their alignment in a real scenario.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 briefly explains our TS solution, used as a reference for the comparison with the WoT. Section 4 examines the similarities and the differences between WoT and TS, and defines how to map both worlds. Section 5 outlines a scenario where both solutions could be useful. Finally, Section 6 concludes and outlines future work.

2. RELATED WORK

There are two main architectures to enable the cooperation between objects within IoT: *service oriented* (SOA) and *resource oriented* architectures (ROA). Both architectural approaches, represented by WS-* and REST styles, were adopted from the World Wide Web area, where their advantages and disadvantages have been extensively discussed [17]. ROA is mainly a simpler decoupled solution which is centred on sharing and managing data resources, while SOA enables the encapsulation of functionalities offered as services and their combination in a more synchronous way.

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

DPWS is a SOA incarnation which defines a subset of WS-* to make it suitable for resource-constrained devices. DPWS is claimed to be used in industrial environments. Its most remarkable features are that it proposes: decentralized multicast-based discovery, secure message transmission, subscription and event notifications. Moritz et al. compared DPWS with the REST approach coming to the conclusion that DPWS can be restricted to be fully compatible with the RESTful style, and still covering some missing features of the RESTful approach such as eventing and discovery [13].

Within resource oriented architectures, the Web of Things (WoT) community encourages the use of REST-based solutions embedding web servers in daily objects [9]. This inherits the simplicity of RESTful based services making it easier for developers to build mashups with other common REST services [8]. Ostermaier et al. [15] proposed a framework to facilitate the integration of different blocks of web enabled sensors and actuators. Considering network connected objects and their capabilities as dynamic and continuous source of information brings another interesting possibility, i.e. sharing them through social networks [7].

Another ROA solution is Triple Space (TS). TS comes from the tuplespace coordination model where the interaction between processes is performed by writing and reading data structures in a shared space. In mobile environments, the tuplespace adoption has probed to yield to simpler, cleaner and more reusable designs [4, 3].

Several solutions have been proposed to bring the semantic web in the tuplespace paradigm [14], including those which follow TS paradigm [5] which simply uses RDF triples as shared data structures. This shared information is processable by the machines since it is modelled as instance data of previously agreed semantic ontologies [2]. In TS there are different strategies that have been used to distribute the semantic data through the nodes which form a space, but to the best of our knowledge this kind of solution has never been specifically designed and implemented to use mobile and embedded devices as fully-fledge peers of these spaces and not only as simple clients apart from in our middleware [6]. Doing so, we can define very autonomous processes (see section 4.1.6) even in constrained environments.

To sum up, TS explores a new path which uses coordination primitives inherited from tuplespace model in a shared semantic space. TS shares similarities with other ROA solutions such as WoT, simpler to use than SOA ones [17].

3. A TRIPLE SPACE SOLUTION FOR IOT

Our TS solution, namely OtsoPack and described in detail in [6], proposes a distributed shared semantic space between nodes which join it. Published semantic data is grouped into graphs which are made up of RDF triples. Nodes generally write locally (except when another one has claimed to be responsible of a piece of knowledge) and spread their queries to the rest of the nodes which belong to the space. This approach fits perfectly with very dynamic spaces where the nodes can change of context frequently (e.g. mobile phones) and a node is usually responsible for the information it manages (e.g. a user profile on mobile phones or sensed data in an embedded device).

The most important primitives of our solution are *write*, *query*, *read*and *take*. *Write* adds new knowledge to a space writing together the given triples in a new RDF graph and

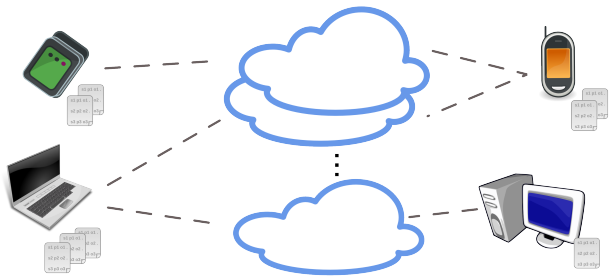

Figure 1: Proposed distributed TS solution.

returning its identifying URI. *Query* returns the triples which match a given template conceiving the space as a whole no matter to which graph they belong to. *Read* and *take* return a whole RDF graph which can be selected by its identifying URI or by a template. In the second case, the graph is returned if it contains at least a triple which matches the template. *Take* differs from *read* because it also subtracts the returned graph from the space.

The template used in these primitives can be as simple as a triple pattern with wildcards (e.g. ?s) or as complex as an SPARQL query, depending on what a node is capable of processing. To make SPARQL queries understandable to all the nodes, the *QueryMultiple* primitive translates them into several wildcard templates. For instance, possible responses for the template *?s rdf:type ismed:Device*, which searches for devices in an environment, are:

```
prefix1:sunspot1 rdf:type ismed:Device
prefix1:xbee5 rdf:type ismed:Device
prefix1:nokia96 rdf:type ismed:Device
```

Two versions of the proposed architecture have been implemented: a fully capable one named OtsoSE and another one more suitable for mobile devices named OtsoME. Besides, information provided by embedded devices has also been fully integrated in the space created by these nodes using the solution described in section 4.2.

4. WOT AND TS

In this section, the strengths and weaknesses of both TS and REST approaches will be discussed and compared. Then, their compatibility is shown by proposing two ways to use WoT solutions inside a TS environment and *viceversa*.

4.1 Comparison

The similarities and differences between WoT and TS which are summarized in the Table 1 are detailed below.

4.1.1 Architecture

Both approaches are based on resource oriented architectures. WoT uses the REST architectural style, where the contents are directly addressable by URLs to manipulate them, and TS creates, removes and modifies RDF triples or graphs, i.e. sets of interlinked triples, in a space formed by peers in a P2P network.

Despite of these similarities, WoT uses a client-server architecture where each client has to know the URL where to address its operations, while TS relies on a distributed shared memory which makes each process within a space completely autonomous. In TS, a node does not need to know where information is stored, just the URI of the space it wants to query. Furthermore, the OtsoPack solution is

Table 1: Comparison between WoT and TS (OtsoPack) resource-oriented approaches

	WoT	TS (OtsoPack)
Architecture	ROA & C/S	ROA & P2P
Communication	HTTP	Jxta
Operations	HTTP verbs	TS primitives
Format	HTML, JSON & XML	RDF (NTriples)
Developer aid	Less expressive data & specific use	Expressiveness & uniform API
Coupling	Low	Very low
Discovery	Bad	Very good
Scalability	Good	Bad
Semantics	Microformats	Full

fully distributed and conceived not to depend on any centralized, previous known, intermediary server.

4.1.2 Communication protocol

Our TS solution currently uses Jxta[1], a language and platform independent *Peer to Peer* protocol, to interconnect the nodes and manage the groups (spaces) where they belong. Unfortunately, the mobile version relies in a Jxta gateway called Rendezvous to propagate the messages to other nodes of the group, making some previous configuration necessary and potentially creating a bottleneck. Anyway, some TS implementations such as TripCom[2] rely on HTTP, and one of our next steps will be the reimplementation of our network communication through HTTP.

Even if it is not mandatory to comply with the RESTful style, WoT usually employs the HTTP protocol as a communication layer because of its simplicity and wide adoption to ensure ample deployment.

4.1.3 Operations

WoT uses HTTP verbs to retrieve, create, modify or delete web resources. Retrieval is done using HTTP GET, creation by means of HTTP POST, modifications with HTTP PUT and removals with HTTP DELETE. Finally, the HTTP OPTIONS command is used to introspectively find out what operations are allowed for any URL.

TS and tuplespace solutions offer different primitives to write, read and delete content. In our solution, the previously explained *write*, *read* and *take* primitives do that. Apart from these primitives, other ones are provided to manage spaces, to claim the manager role for a type of content in a space, event notification or traditional request-reply style service consumption. Anyway, in this paper we are going to focus in the first ones because they are inherent to the TS paradigm.

4.1.4 Format

WoT usually returns a human readable HTML representation for a resource and XML or JSON common web representations to be used in mashups. One of the key mechanisms that WoT inherits from HTTP is *content negotiation*, which enables clients and servers to negotiate the requested and provided representations for any given resource.

TS does not provide this response format negotiation mechanism. So, OtsoPack interchanges data in N-Triples format, which is the most primitive representation of semantic data. In any case, other alternative semantic representations (e.g. RDF/XML, OWL/XML, Turtle or N3) could be easily adopted as they basically describe the same RDF triples under different textual wrappers.

4.1.5 Developer aid

In ROA an agreement about the data being exchanged and how it can be accessed is needed to develop applications. On the one hand, WoT can define machine processable data using XML or JSON, but thanks to the Semantic Web TS can go one step further formally defining them and making them also "machine understandable" (i.e. it is capable of inferring new knowledge and make high-level queries over it).

On the other hand, both WoT and TS offer some operations and interfaces to access them. Whereas in TS this interface is the same in each node and it is closely related with TS operations and resources, in WoT it changes in each application making a previous learning process necessary to use them.

4.1.6 Coupling

The REST style describes loosely coupled services due to the platform independent, asynchronous and few self describing messages [16]. In a very similar manner, TS offers different kind of autonomies [12]: *time autonomy* (because of its asynchronous nature), *location autonomy* (information providers and consumers are independent from where the data is stored), *reference autonomy* (nodes do not need to know each other) and *data schema autonomy* (it follows the RDF specification making it independent of nodes internal data schema).

In spite of the outlined decoupling nature of both approaches, the data definition can be considered a coupling mode indeed. While in WoT each resource defines its own data formats and contents themselves, in TS the ontology in which the semantic concepts are described must be known by each part of a distributed application to effectively cooperate among them.

4.1.7 Discovery

One of the main drawbacks in WoT is the lack of a discovery mechanism for new objects and the data they provide. Even when this data can be linked in each object response (using HATEOAS) and microformats are sometimes included to ease the search-ability of these objects by search engines, it is difficult for an object which may change of location and context to be referred. Thus, WoT may have a tendency to create isolated islands of data. Several workarounds have been proposed to overcome this limitation, such as using a central repository[3], a framework which uses federated repositories responsible for different administrative domains [18] or making each connected sensor announce itself to let an intermediary know its presence [11].

TS provides a transparent data level discovery mechanism to the user since when a node joins a space (i.e. a group), its data become queriable by any other node joint to this space. In OtsoPack the space management and communication relies on Jxta, a protocol which can be used at any

[1]http://jxta.kenai.com/

[2]TripCom (IST-4-027324-STP, www.tripcom.org)

[3]http://www.pachube.com/

level (even if we have mainly used it on local environments using Jxta's discovery layer's capabilities).

4.1.8 Scalability

The scalability of WoT is argued to be well proved since the World Wide Web is the most practical scalable system. As many objects as necessary could be added to WWW without making it worse.

Although Jxta scalability has been discussed and addressed in several works [1], since our solution is fully decentralized and uses flooding (each query is spread to the rest of the peers which belong to a space), its scalability is expected to be poor. To overcome this limitation, we are working on implementing Semantic Overlay Networks (SON) to let the objects automatically divide the spaces into smaller ones and address queries just to the appropriate nodes which conform a semantically related space.

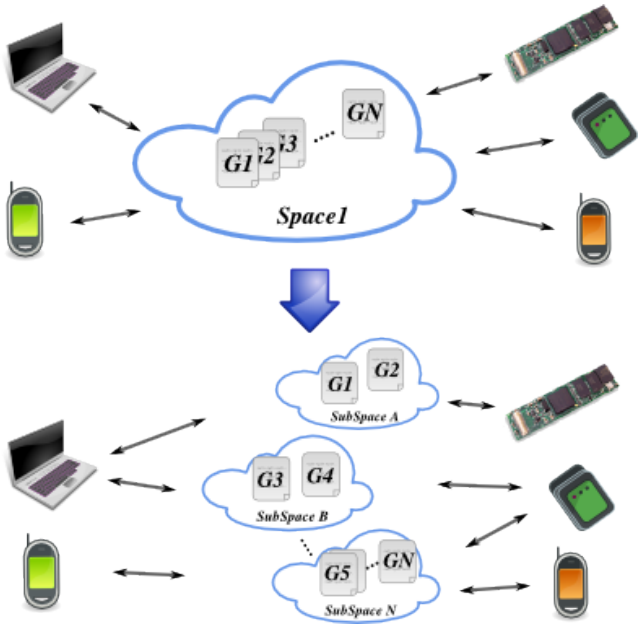

Figure 2: Proposed distributed SON-based solution.

As can be appreciated in Figure 2, the proposed solution aims to split the spaces up into smaller subspaces. To do so, semantically similar contents are placed together in the same subspaces. E.g. *subspaceA* contains *graph1* and *graph2* which describe "light" related knowledge. In the meantime, the nodes from the original space (*space1*) can organize themselves to create another *subspaceB* containing the relevant data for describing "mobile phones".

4.1.9 Semantics

On the one hand, WoT uses predefined microformats to embed semantic information into human readable pages. Doing so, the search process performed by search engines is enhanced [9]. On the other hand, TS allows the usage of full semantics, more expressive than microformats, using RDF as a base. This makes TS capable of using standard query languages such as SPARQL and it becomes also independent of third parties' products to search data.

4.2 Using WoT in a TS solution

To demonstrate the complete compatibility between TS and WoT approaches, we first used a WoT solution in a TS node. We used a gateway [10] which provides access to sensors and actuators through RESTful services. To adapt it to the TS paradigm, we added to it a new data representation using a set of semantic triples (in this first approach, the solution is dependent on the ontology of the scenario).

In OtsoPack, each node is mainly made up of two parts: the network layer and the data access layer. While the first one has the responsibility of keep a node communicated with the rest of the nodes of the space, the second one stores the triples managed by this node. In OtsoSE we have replaced this data access layer to obtain the semantic information from the gateway instead of from a semantic repository. Doing so, the TS primitives addressed to the gateway are translated into an HTTP request as summarized in Table 2.

The Read primitive has two different types of implementations. In the most basic one, the graph is identified by a URI which in this case coincides with the URL of the service that returns it (for example, in the case of temperature sensor http://node/sunspots/*SpotName*/sensors/temperature/). In the second implementation, the template is passed in as a query string parameter for the GET command issued to a specific URL and the gateway checks all the graphs to facilitate a response. The Query operation has a similar behaviour. The Write parses the contents of the triples extracting values and makes a POST request with them over a particular actuation service URL to change its state.

4.3 Making TS nodes part of the WoT

In this section, a proposal to make any TS node WoT compliant is explained. To do that access to a TS should be provided through a RESTful service (as shown in Table 3). As in TS, spaces, graphs, subjects, predicates and objects are identified by URIs, in order not to make the requesting URLs too long, each node should provide a prefix mechanism to enable the URI shortening at *http://nodeuri/prefixes/*. This node will return all the prefixes used by this node, so they can be used inside any URL by simply using a name followed by ":" and the last part of the URI.

To see the graphs available in a concrete node, *http://nodeuri/spaceuri/[graphs]* could be accessed. To access each graph in a space, no matter if it is stored by the node responding to the HTTP request or not, we could access *http://nodeuri/spaceuri/graphs/[graphuri]*. It will be internally translated into a *read*primitive. To locate a graph giving a *template* the accessed URL will be *http://nodeuri/spaceuri/graphs/[template]*. The HTTP DELETE verb should return the graph and delete it from the node where it was stored.

We propose specifying first a subject *subject/[subj-uri]/* and concatenate */predicate/[pred-uri]/* and *object/[obj-val]* if needed to specify a *template*. The order should be this, but any of them could be optional to express that any value could be ok (wildcard option). To express a *?s ?p ?o* .-like template (any triple matches it), the URL ended by *any* could be used.

To perform a *queryhttp://node/spaceuri/query/[template]*-like URL should be accessed and to write a new graph the user should make a HTTP POST request to *http://nodeuri/spaceuri/graphs/* obtaining the new graph's URI as response.

Table 2: Mappings between OtsoPack's primitives and HTTP requests addressed to a WoT solution.

TS primitive	HTTP request
read(spaceURI,[graphURI])	HTTP GET over [graphURI]
read(spaceURI,[template])	HTTP GET over http://gateway/read?template="[template]"
query(spaceURI,[template])	HTTP GET over http://gateway/query?template="[template]"
write(spaceURI,[template])	HTTP PUT over http://gateway/sunspots/[SpotName]/leds/led[0-6] Parameters: • switch=[true/false] • redColor=[0-255] • blueColor=[0-255] • greenColor=[0-255]

Table 3: Examples of REST access to TS (*sp:ex* is a space URI, *sp:gr1* is a graph URI and templates are expressed between quotes)

HTTP request	URL	Returns
GET	http://nodeuri/prefixes	The list of prefixes used by the node
GET	http://nodeuri/prefixes/sp	The URI that "sp" prefix represents
POST	http://nodeuri/prefixes (parameters: *URI & prefix name*)	Add a new prefix
GET	http://nodeuri/sp:ex/query/any	`query(sp:ex,"?s ?p ?o ."): triples`
POST	http://nodeuri/sp:ex/graphs (parameter: *triples*)	`write(sp:ex,triples): URI`
GET	http://nodeuri/sp:ex/graphs	The list of graphs stored in this node
GET	http://nodeuri/sp:ex/graphs/sp:gr1	`read(sp:ex,sp:gr1): triples`
GET	http://nodeuri/sp:ex/graphs/subject/sp:s1	`read(sp:ex,"<sp:s1> ?p ?o ."): triples`
DELETE	http://nodeuri/sp:ex/graphs/sp:gr1	`take(sp:ex,sp:gr1): triples`
DELETE	http://nodeuri/sp:ex/graphs/object/sp:o1	`take(sp:ex,"?s ?p <sp:o1> ."): triples`

5. DISCUSSION

In the previous section, the two-way compatibility between the TS and WoT approaches has been described. As has been shown, WoT is good due to its scalability and usage of web standards, which makes it easy to be understood by potential developers (which encourages its usage). Our TS solution, on the other hand, provides good local discovery of new resources and their information and allows more expressive semantic data definition which leads to more expressive and sophisticated queries. Taking this into account, a scenario which takes advantage of the potential combination of these approaches has been sketched (see Figure 3).

As the TS solution allows to easily deploy and share data among network-connected objects in local environments (thanks to the dynamicism of *P2P* architecture and the data level *discovery*), a user could create two spaces both at home and at his workspace populated by both TS native nodes or wrapped web enabled objects (as explained in section 4.2).

Both places will provide a REST API through a nearly always turned on machine. Doing so, the *scalability* of WoT systems is inherited and at the same time compatibility with WoT solutions and general REST services is ensured. This would allow to this user or another skilful, and authorized, developer to make a mash-up to create a common management interface to provide access to all the sensors on both spaces. This control panel could be enriched with external services such as the weather in each physical location or an WoT device, and it will provide a rich query interface to take advantage of the knowledge held in the underlying TS.

The user and his flat-mates, will have a semantic profile created using a wizard and hosted in their mobile phones. Whenever one of them enters their home, the mobile will connect to the home TS (e.g. *ts://aitor/home*) acting as a new node which attempts to adapt the environment (home) to its owner's preferences. For instance, it will try to adjust the temperature according to the user's preferences writing the temperature preference in the TS shared with the network-connected air-conditioning system. On the other hand, the mobile phone could easily monitor the home both querying the TS space directly whilst at home or through the REST access both at home when it is away.

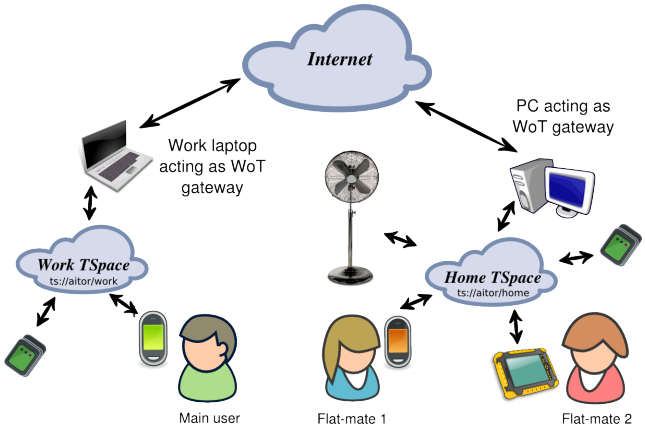

Figure 3: Proposed hybrid scenario.

6. CONCLUSIONS

This paper compares two different resource oriented approaches for the Internet of Things: Web of Things and Triple Space. WoT seems to scale in a better way thanks to the underlying HTTP protocol, while TS performs the discovery process among locally available network-connected objects in a seamless way. The first one is more human oriented and the second one relies on the Semantic Web capabilities to exchange a machine processable data. Furthermore, the second one is ideal to easily configure intranets of network connected objects whilst the first one can easily bridge those intranets configuring global multi-site IoT ecosystems.

We deem that both approaches can win much from their combination since the weaknesses of one are outweighed by the strengths of the other. Hence, they can be combined to offer a more scalable, machine and human processable solution that offers better cooperation possibilities among internet-connected objects and thus aid users in their daily activities. As a simple proof of this hypothesis, a scenario employing WoT to export each space data to the outer world and TS to enable seamless and automatic configuration of heterogeneous devices on local networks has been presented.

For our future work, we are planning to implement TS over HTTP to achieve a hybrid WoT and TS solution. In addition, we plan to centre our effort both on security, particularly authorization, issues and on improving the performance of our solution using semantic overlay networks to enable nodes auto-organization. Besides, the inclusion of a semantic reasoner to allow more intelligent coordination among distinct networked objects targeted towards better pleasing user needs will be considered. Finally, a performance analysis in heavily populated simulated scenarios should be performed.

7. ACKNOWLEDGMENTS

This project has been financed under grant PC2008-28 A by the Department of Education, Universities and Research of the Basque Government for the period 2008-10.

8. REFERENCES

[1] G. Antoniu, L. Cudennec, M. Jan, and M. Duigou. Performance scalability of the JXTA P2P framework. In *2007 IEEE International Parallel and Distributed Processing Symposium*, page 109, 2007.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. *The Semantic Web*, volume 284. 2001.

[3] P. Costa, L. Mottola, A. Murphy, and G. Picco. Programming wireless sensor networks with the teeny lime middleware. *Middleware 2007*, pages 429–449, 2007.

[4] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco. Tinylime: Bridging mobile and sensor networks through middleware. 2005.

[5] D. Fensel. Triple-space computing: Semantic Web Services based on persistent publication of information. *Intelligence in Communication Systems*, pages 43–53, 2004.

[6] A. Gómez-Goiri, M. Emaldi-Manrique, and D. López-de-Ipiña. A semantic resource oriented middleware for pervasive environments. *CEPIS UPGRADE journal*, page 6, Feb. 2011.

[7] D. Guinard, M. Fischer, and V. Trifa. Sharing using social networks in a composable web of things. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 702–707, 2010.

[8] D. Guinard and V. Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of International WWW Conferences*, Madrid, Spain, 2009.

[9] D. Guinard, V. Trifa, F. Mattern, and E. Wilde. From the internet of things to the web of things: Resource oriented architecture and best practices. In *Architecting the Internet of Things*. Springer, May 2011.

[10] D. Guinard, V. Trifa, and E. Wilde. A resource oriented architecture for the web of things. In *Proceedings of Internet of Things 2010 International Conference (IoT 2010)*, Tokyo, Japan, Nov. 2010.

[11] A. Kamilaris, V. Trifa, and A. Pitsillides. The smart home meets the web of things. *International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC), Special issue on The Smart Digital Home*, 2010.

[12] R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel. WWW or What is Wrong with Web services. In *Web Services, 2005. ECOWS 2005. Third IEEE European Conference on*, page 9. IEEE, 2006.

[13] G. Moritz, E. Zeeb, S. Pruter, F. Golatowski, D. Timmermann, and R. Stoll. Devices profile for web services and the REST. In *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, pages 584–591, 2010.

[14] L. J. Nixon, E. Simperl, R. Krummenacher, and F. Martin-Recuerda. Tuplespace-based computing for the semantic web: a survey of the state-of-the-art. *The Knowledge Engineering Review*, 23(02):181–212, 2008.

[15] B. Ostermaier, F. Schlup, and K. Romer. WebPlug: a framework for the web of things. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 690–695, 2010.

[16] C. Pautasso and E. Wilde. Why is the web loosely coupled?: a multi-faceted metric for service design. In *Proceedings of the 18th International Conference on World Wide Web*, pages 911–920, 2009.

[17] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceeding of the 17th International Conference on World Wide Web*, pages 805–814, 2008.

[18] V. Stirbu. Towards a RESTful plug and play experience in the web of things. In *The IEEE International Conference on Semantic Computing*, pages 512–517, 2008.