A Meta-Model for the Web of Things

A. Ruppen & J. Pasquier

Internal working paper no 13-03

June 2013

UNIVERSITAS FRIBURGENSIS

UNIVERSITÉ DE FRIBOURG / DEPARTMENT OF INFORMATICS

A Meta-Model for the Web of Things

Andreas Ruppen and Jacques Pasquier

Software Engineering Group,
University of Fribourg,
Fribourg, Switzerland
{andreas.ruppen,jacques.pasquier}@unifr.ch
http://diuf.unifr.ch/softeng

Abstract. The Internet of Things is a heterogeneous place. No restrictions are made regarding protocols and interfaces. The Web of Things (WoT) tries to bring some standards through RESTful web-services. Yet, there exists no globally accepted meta-model with a well defined set of supporting tools, which could help software architects and developers design complex applications embedded in this new ecosystem of smart-devices interacting with web-services.
In this paper, we start by identifying the different parts involved in WoT applications. Then we present a meta-model including them all in a coherent way. Finally, we conclude by illustrating the usefulness and the applicability of the meta-model through an illustrating example.

1 Introduction

Following Moore's law [1], transistor count doubles each year. As a side effect costs for a given device (like a CPU) become cheaper each year, even if the decrease of the price might not be as strong as the increase in transistors. These two factors lead to the appearance of low-cost hardware with limited communication capabilities. The Internet of Things (IoT) is an Internet where the participants are Things [2]. A Thing is mainly a piece of hardware, like a sensor, having some communication capabilities. Thus Moore's law is an enabling factor for the Internet of Things. The technological advances over the past decade lead to really cheap devices, full with sensors, having a decent autonomy and connectivity like the Arduino boards [1] or the Sun SPOT [2].

Over the past few years, researchers have shown a big interest in connected Things. Projects like Cooltown [3] emerged. All these research efforts led to different approaches on how to build smart-devices, the foundation of the Internet of Things (IoT). More recently, the Web of Things (WoT) appeared [4]. The WoT stands on top of the IoT by adding a standardized layer, connecting individual smart-devices together. Instead of defining new architectures for each scenario, which leads to a fragmented IoT, it uses the web as a fully fledged application protocol. An enabling architecture consists in using RESTful web

[1] http://www.arduino.cc/
[2] http://www.sunspotworld.com/

services. Indeed, on the one hand they present several advantages over their WS-* counterparts [5] and, on the other hand, they represent today an accepted way of designing architectures for smart things [4, 6]. Another trend to take into consideration is the shift from using single raw services towards integrating them with computationally complex processes [7, 8].

In this paper, we identify the different parts involved in WoT applications. Once these parts defined, we discuss a meta-model for the WoT. This meta-model shall bring down the complexity of WoT applications and lower the entry barrier for developers, helping them to make the right design decisions. Another of its benefits will be to define a clear and common vocabulary in the field of smart devices and WoT applications.

The rest of the paper is structured the following way: we first motivate the need for a meta-model by examining the growth of complexity in WoT applications. Section 3 constitutes then the heart of the paper and is divided into five parts: (1) a short introduction to the field of meta-modeling; (2) identification of a WoT applications key concepts; (3) illustration through a simple example; (4) formalization of the meta-model; (5) reassessment of the illustrating example through this new formalization. We then discuss some related work and open the discussion about future work.

2 Motivation

Recently, the WoT is shifting from raw devices to services. In [7] the authors propose a marketplace for algorithms and their easy integration into the WoT. They motivate the need for such linked algorithms by the statement that values produced by smart devices are useless if they are not aggregated and augmented in some way. It is foreseeable that in a near future, more devices will be connected to the Internet than there will be humans. This evolution leads to a real flood of information which has to be tackled somehow. Through the mechanism of linked algorithms they aim to "distill" the information. Another trend is the shift from simple mashup applications to more complex scenarios involving decomposable and delayed services [8] and even business process[9, 10]. Accordingly, we are drifting from classical WoT scenarios made of raw smart-devices to mashup applications including sensors, actuators and tags as well as computational services. These services might include algorithms, market place applications and business processes.

As the landscape of application scenarios gets broader, it becomes important to share a common base. This base not only involves a clear vocabulary, but also guidelines on how smart devices and resources should be built. Having a common meta-model will later allow to automatically create the interfaces of such services. Convention over configuration or coding by convention is an accepted design paradigm and its popularity is increasing. A prominent example is the Maven [3] project management tool. By accepting the Maven conventions on

[3] http://maven.apache.org/

the structure of a project, the system can handle much of the aspects of project management out of the box. The same can be true for the WoT. By using a proper meta-model for all concrete future models, we can deliver a certain amount of homogeneity to the WoT without abandoning its loose coupling. Having a meta-model, from which all models derive, will later facilitate not only the direct usage of WoT services by humans, but also their automatic composition. Furthermore, code skeletons can be derived from the models and greatly increase the speed of development.

3 The Meta-Model

3.1 Definition

Meta-modelling is a three layered approach. On the lowest layer sits the data or the subjects under study (SUS). These are concrete endeavors. One layer above are the models. They are used to formally describe SUS by abstracting from the concrete details. By that, a model describes what is shared by a class of SUS. Still one layer above are the meta-models. They aim to abstract models and describe what is common in a group of models.

There exist models for different types of situations. A prominent example are UML models. They can design various situation in software engineering. An instance of such a model would, for example, be a class diagram showing the structure of some code. Yet all these models should share a common base and a common vocabulary. This makes it easier to read and compare models from different sources. Furthermore, this also allows the automatic creation of models out of code and vice-versa. As a model aims to describe common structures for a class of SUS, which can then be translated into different instances of code, a meta-model defines the common structures and available elements of all models [11].

Restricting the application of our meta-model to the WoT allows us to concentrate on the specificities of smart-devices without caring about device electronics. How RESTful web-service communicates with a given hardware is very specific and also manufacturer dependent. For this reason it should be omitted from the meta-model. Thus, we are concentrating on resources oriented architectures, but from the perspective of the Web of Things.

3.2 A Component Based View of the WoT

The bricks forming the WoT are smart-devices. A smart-device has some inner structure made of bare sensors, actuators, tags and/or algorithms and a well defined RESTful interface over which it can communicate. Accordingly, a smart-device is twofold: it is composed of some electronics (the device aspect) and some communication capabilities (the smart aspect). Components are twofold too: they have some inner logic and a well defined interface with the outer world. How the inner logic is structured is shielded form the users. Therefore, components

are like a black box providing some service, without knowing how the operations are performed. By that, smart-devices can be seen as components. Later, we will see that each component is an entity of interest. Seeing a component as an entity of interest also provides a rough guideline on how to separate them. Each one is representing one entity. The combination of such components is used to build mashup applications. New applications emerge out of different combinations of already existing parts. Since each component exposes a RESTful interface, their combination is easy. Moreover, mashup applications are what a user generally deals with and what users can see from the WoT. How they can be combined has already been studied in other papers (for example [12]) and is out of the scope of the present one.

Instead of going on step up, from components to mashup applications, it is also possible to go one step down and look on the underlying structure of such a component. Each component has to be structured following some design principles and patterns. The meta-model takes care of this structure. It names the main elements composing a component and describes their relations. Additionally, it also takes care of the RESTful interface by providing an adequate structure to the latter. By that, the meta-model simultaneously takes care of the inner guts of a component and its outer interface. In practice, each component will be described by a model, which is an instance of the meta-model.

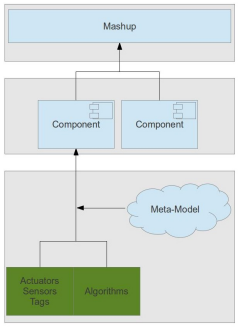

Fig. 1. 3 Layers of WoT Applications

This leaves us with a three layered approach, each of the identified elements above being one of the layers. From bottom up these are: (1) the *meta-model* structuring the (2) *components* which, when combined form (3) *mashup applications*. These three layers and their relation are resumed on Figure 1. If the bottom two layers follow clearly defined patterns, the creation of mashup applications is greatly simplified.

3.3 Illustrating Example - Smart Light Bulb

Before discussing the notion of components and its associated meta-model, let us have a look at a small example to better illustrate the 3 layers introduced above. We will come back to this example in Section 3.5 and model it properly accordingly to the meta-model. Let us consider a standard light bulb, as they are used everywhere. It has a state which is either on or off. Moreover it can be switched on or off. The state is changed with a button switch. To make this light bulb a smart device, it has to accept commands over a well defined interface connected to the Web. Furthermore, the state also needs to be reflected on the Web so that it can be exploited by other applications relying on it. One possible way to reflect this state to the virtual world, is to add a light sensor measuring the available light and detecting whether the bulb is switched on or off. At this stage we can already place the three layers in context: (1) Mashup applications will rely on the interface provided by the smart bulb to build rich application scenarios like home automation systems. (2) The light bulb is one of the components implicated in such a scenario. It represents one of the different key players of a home automation system, among others like a heater, roller blinds etc. Each component is an entity, representing some interest to a final application. (3) Finally, each component needs to be structured following standard patterns and models. Therefore, the meta-model is used to describe its inner parts and outer interface. Throughout the rest of this paper, we will concentrate on this last part, the meta-model and ignore the layer above. Once the meta-model is fixed, we can then build upon it and examine models for components and mashups but this discussion is out of the scope of this paper.

3.4 Formalization

Starting from the considerations above, suppose that the space is already divided into components, each one representing an entity of interest for future scenarios. We further make the assumption that each component is part of the WoT and thus, exposes a RESTful interface over which it can communicate. Since this interface is the contract between the client and the service, it is worth spending some time and thinking about how to structure it. Even though a client can discover connected resources by following links, this interface should be stable over time.

Additionally, the meta-model is also able to model applications not involving directly smart-devices, such as marketplace applications like the one in[7] or other pure algorithms enhancing the power of the WoT. As motivated in Section 2, such applications bring an added value to the WoT. Therefore, we can divide the available service into two categories:

1. *Physical services* have an associated device and make the latter available over a RESTful interface.
2. *Virtual services* are not tied to a specific device. They provide generic algorithms of value to the WoT over a RESTful interface.

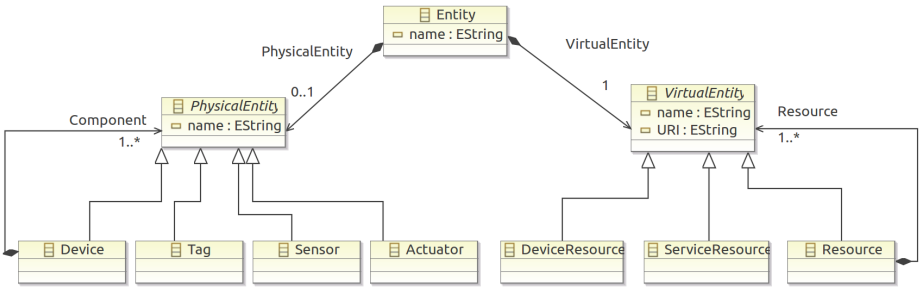

Fig. 2. Meta-Model for WoT Components

Even if the second category is, at a first glance not related to typical WoT scenarios, we argue, for the arguments given before, that they have to be treated as first class citizens in our model.

Figure 2 shows the elements composing the meta-model. The *Entity* is a global concept embracing a physical and/or a virtual part. Some prefer calling it "entity of interest" [13] to underline that this is what the user is interested in. An *Entity* can embrace anything somebody is concerned about. This includes objects like books or doors, but also creatures like cows or humans. An *Entity* can be composed of a *PhysicalEntity* and must have a *VirtualEntity*. The *PhysicalEntity* represents the physical part of the *Entity*. It can measure, act on or stand for this physical part. An instance of such a *PhysicalEntity* would for example be a sensor measuring a physical property. The *VirtualEntity* stands for the virtual part of an *Entity*. It can also measure, act on or stand for an *Entity* but does this in the virtual world. As such a *VirtualEntity* would hold the value of a temperature sensor attached to the *Entity*. By that, the *VirtualEntity* reflects the character of the *PhysicalEntity* in the virtual world. Therefore, the two are highly coupled. Acting in the physical world should lead to the same results as acting on the virtual side of an *Entity*. WoT applications imply that for each *PhysicalEntity* there exists a virtual counterpart, the *VirtualEntity*, both representing the same. By that, for each sensor in the real world, there exists an instance of *VirtualEntity* holding the same information. As a result, each *PhysicalEntity* always relates to a *VirtualEntity*. The inverse, however, is not always true. This is why we speak of a one-to-one mapping between the former and the latter. A *VirtualEntity* can exist without the presence of a physical counterpart. This is especially true for virtual services, since they are not associated with any device directly. Thus, the two sets of services identified above are both present in the meta model. For of physical services, there is a one-to-one mapping between *Physical-* and *VirtualEntity* whereas for virtual services the model has no instance of *PhysicalEntity*. This also explains the different cardinalities on the two associations of Figure 2. To illustrate this three concepts, let us look at an example: a door can be opened or closed. Likewise, a door has two states: open and closed. Yet, such a door is not smart, in the sense that it has no virtual counterpart. Although, by adding a sensor, it can be made smart. Thanks

to this sensor, the state can be propagated to the virtual counterpart of the world. Accordingly, the door is an instance of the concept *Entity*. It has an associated instance of *PhysicalEntity*, the sensor and an instance of *VirtualEntity* representing the door in the virtual world.

Besides the already discussed elements, Figure 2 includes further interesting concepts. The *PhysicalEntity* decomposes into multiple concrete classes. According to S. Haller [13], smart-devices can be divided in three categories: (1) Sensors are measuring a physical property, (2) Actuators are acting on the physical environment and (3) Tags are used for identifying objects. Following the composite pattern, a *Device* is a *PhysicalEntity* without being a *Tag*, *Sensor* or *Actuator*. However, it can hold several of them and they form the leafs of the composite pattern. To better motivate this choice, let us look again at the door example. In addition to the open/closed state, it would be nice to know whether the door is locked or unlocked. To gather this new information, another sensor is attached to the door. Accordingly, the *PhysicalEntity* is now composed of two parts: a sensor measuring the open/closed state of the door and another sensor measuring the (un-)locked state. Thus, the *Entity* is now composed of a compound of devices, each sensor providing another information. This composition is reflected in the meta-model under the assumption that both sensors are on a same device. On a high level, this assumption has no influence on the real wiring. The two sensors can still be separated physically, as long as they relate to the same *Entity*. We can therefore assume, without loss of generality, that the sensors are glued on a same device, thus following the composite pattern. To conclude the example, each of them need an associated *VirtualEntity* representing them on the virtual side.

As for the *PhysicalEntity*, the *VirtualEntity* also decomposes following the composite pattern for mainly two reasons: (1) there exist a one-to-one mapping between the physical and the virtual world and (2) URIs are hierarchical. To motivate the first reason, let us come back to the door example. The sensors still need to be mapped to resources to be reachable over the web. To model this in the *VirtualEntity*, the virtual door needs a resource representing it. This resource holds information like the color or the dimensions. It decomposes into two sub-resources, one holding the open/closed state, and another reflecting the (un-)locked state. As for the second reason, resources decomposing into sub-resource is not a new concept. It is commonly accepted that a resource can be hierarchically decomposed [14] and so does the *VirtualEntity*. As shown on Figure 2, the leafs of the composite pattern are twofold; a *DeviceResource* refers to a physical device and is needed to capture WoT specific requirements. A *ServiceResource* is not linked to any device and refer to pure algorithms embedded in the WoT. From a client's perspective, this difference will probably not be visible, the interface being the same. The compound structure is called *Resource* and is necessary at least during the modeling phase. Whether it is later translated into code or not depends on the situation. Sometimes, this list element is mandatory for the interface, whereas in other situations it is better to hide it [14].

This meta-model has been formalized in EMF [15]. The EMF syntax is very simple and mostly concentrates on the aspects discussed above. Additionally, the EMF contains some fields not discussed in details here, but they can be discovered in Figure 2 and Listing 1.1 and are self-explaining. Thanks to the tools available for meta-modeling in Eclipse, a new plugin was generated and, once installed in Eclipse, allows to create new projects based on the meta-model. With the aid of the graphical tools packaged with Eclipse it is easy to instantiate now a WoT model of an endeavor. Furthermore, this model can be translated into executable Java code based on the Jersey library. This transformation is done in two steps by parsing the generated XML file of the model. We will see the usage of this tools in Section 3.5.

```
package wot;

class Entity{
        attr String name;
        val PhysicalEntity[1] PhysicalEntity;
        val VirtualEntity[0..1] VirtualEntity;
}

abstract class PhysicalEntity{
        attr String name;
}

class Device extends PhysicalEntity{
    val PhysicalEntity[+] Component;
}

class Tag extends PhysicalEntity{

}

class Sensor extends PhysicalEntity{

}

class Actutator extends PhysicalEntity{

}

abstract class VirtualEntity{
        attr String name;
        attr String URI;
}

class Resource extends VirtualEntity{
    val VirtualEntity[+] Resource;
}

class DeviceResource extends VirtualEntity{

}

class ServiceResource extends VirtualEntity{

}
```

Listing 1.1. Formalized EMF Model

3.5 Example Revisited - Smart Light Bulb

Let us consider again the smart light bulb of Section 3.3. We have seen that to make a light bulb smart, it is necessary to add a light sensor, measuring the available light and therefore detecting the current state. Additionally, the light bulb has an actuator which can switch its state. This description can then be modeled with an instance of the meta-model. Figure 3 shows the resulting model. Clearly the Light Bulb is an instance of *Entity* and represents what we are interested in for this use-case. From the description above, we know that we have

to add two elements on the physical side, a light sensor which is a realization of *Sensor* and the on/off switch which is a realization of *Actuator*. Both are instances of *PhysicalEntity* and are grouped together to form one instance of *Device*. The result of this process can be seen in the upper left part of Figure 3. To embed the light bulb in the WoT, and as specified by the meta-model, we still need to add an instance of *VirtualEntity* and give it a virtual representation. Since on the physical side there are two devices, this same structure needs to be reflected in the virtual world. Therefore, the instance of *VirtualEntity* is made of two instances of *DeviceResource*, one representing the light sensor and the other the button switch. To be consistent with the model, both are grouped together in one top-level resource, representing the light bulb. The result of this process can be seen in the lower left part of Figure 3.

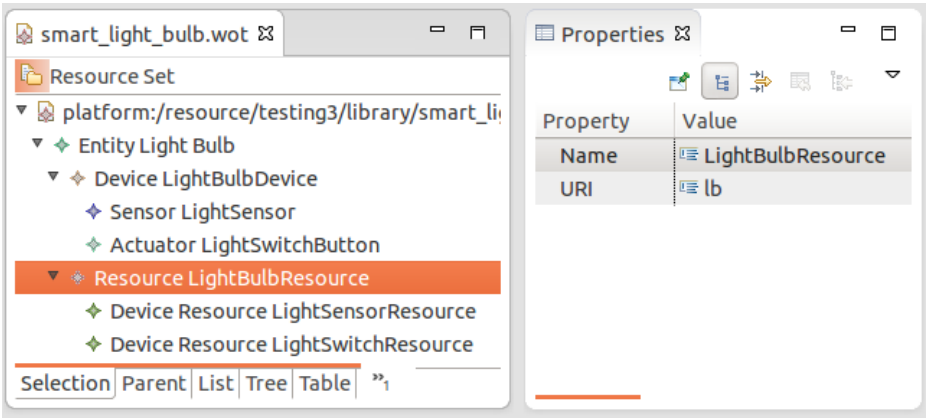

Fig. 3. Smart Light Bulb Model

The model of Figure 3 can then be compiled to Java code using Jersey annotations to define the resources. This compilation phase is done in two steps, the first step producing the WADL file of Listing 1.2. In a second step the WADL definitions are then translated to Jersey annotated Java code. As shown in Listing 1.2 the parts describing the physical world have disappeared from the description. They will be added back later when implementing the body of each Java class representing a resource. Only the virtual parts stay in the WADL file. Furthermore, each instance of *VirtualEntity* is translated into a resource, respecting the defined hierarchy between resources. As such, the LightBulbResource, being an instance of *Resource* decomposes into two sub-resources, both instance of *DeviceResource*. The first representing the light sensor and the second representing the button switch.

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://wadl.dev.java.net/2009/02" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance" xsi:schemaLocation="http://wadl.dev.java.net/2009/02
    http://www.w3.org/Submission/wadl/wadl.xsd">
      <grammars>
```

```
            <include href="smart_lb.xsd"/>
        </grammars>
        <resources base="http://localhost:9090/mySystem/">
            <resource id="com.example.lightbulb.resources.LightBulbResource" path
                ="/lightbulb"/>
            <resource id="com.example.lightbulb.resources.LightSensorResource"
                path="/lightbulb/state" />
            <resource id="com.example.lightbulb.resources.LightSwitchResource"
                path="/lightbulb/switch" />
        </resources>
</application>
```

Listing 1.2. application.wadl extract of the Smart Light Bulb

4 Related Work

Several approaches have been undertaken to introduce some standards to the vocabulary and the structure of the IoT. S. Haller proposed a first draft of a reference-model for the IoT [13]. In his paper he identifies the key players in systems involving smart-devices and their fundamental relations. However, by its heterogeneity, it is difficult to provide a model broad enough to consider most situations and still precisely cut for the IoT. Such a model has to take into consideration simultaneously low-level hardware specifications and designs but also software components and conceptual artifacts. This dilemma can be clearly seen in the reference-model provided in [13] where almost each concept is a composition of itself and almost all connections are of type "many".

With the effort for integrating more and more computational resources comes new challenges. In [7] the authors propose a computational space for the Web of Things. This space can be seen like a market place for applications which embeds in the WoT. The authors state that creating a "smart-world requires the distilling of more abstract, higher-level information from it". Algorithms can be integrated to abstract from the raw-data and gain additional value. This integration is done by using linked algorithms. In [16] and [8] the authors show that the value of the WoT can be greatly augmented by combining WoT like services with services not relying on any physical device. They motivate this conclusion with the study of an eHealth scenario and show that real-world scenario implies both, smart-devices and services.

Whereas, it is easy to consume the value generated by a sensor, basically one simple request, it can be challenging to pipe values between different resources. The simplest situation illustrating this problem is a sensor giving access to a temperature reading in degrees Fahrenheit and a process responsible for augmenting and decreasing the heating. If the process also uses Fahrenheit, then everything works as it should. However, what happens if the process works with degrees Celsius? Even worse, how a client knows that a sensor-value is of a given type. To address such problems a resource can be annotated with ontologies. In [17], the authors propose to annotate the resources with a microformat [4], hREST. Microformats are defined as an extension to HTML. They map the hierarchy found in an HTML page into another hierarchy made of objects and associated

[4] microformats.org

properties. The presented microformat allows to add information to the already existing HTML description of the service, making it machine readable. This information can then be extracted into an RDF-graph allowing further treatment. Yet, they introduce another level of complexity. The authors of [18] propose an ontology for the IoT. This ontology should also guide developers when building IoT systems. They state that the heterogeneous IoT need a homogeneous way of integration of the data they provide.

Yet another approach to bring a common structure to the WoT are the semantic web and Triple Spaces [19]. Its roots goes back to parallel processing and to Linda, developed at Yale University. Triple Space Computing (TSC) combines the communication model established from space-based computing and enhances it with semantics. This approach is not so far away from RESTful web-services. One of the Triple Space Computing requirements is "web-like communication". This is not surprising, knowing that Tim Berners Lee was involved in the discussions about the relationship of space-computing and semantics. Additionally, TSC is based on the work of R. Fielding [20] and takes REST as an endorsing technology. It has been proven that TSC can be applied to the WoT [21].

A commonly accepted way to model and compose services are component based architectures [22]. In such architectures, each components encapsulates a given functionality. To the outside, components look like a black box. They have a clearly defined interface over which interactions with them are possible. Examples for such components are web-services. They have a clearly defined interface to the outside, but their business logic is encapsulated and hidden from the user. Component based architectures imply a loose coupling. Each component can be exchanged with another one (even at run-time), as long as the successor keeps the same interface. Whereas components can take the form of objects in classical object-oriented programming, they can also take the form of resources in the WoT. However component based architectures do not help regarding the inner design of a component.

For these reasons we are looking for a simple, yet complete meta-model, capable of modeling smart-devices but also computational resources. Since RESTful web-services are an accepted standard for connecting smart things, the presented meta-model only applies to the Web of Things. As far as it only concerns RESTful web-services the paper [23] already proposes a deep reflection about a meta-model for RESTful web-services and sketches up such a model based on Ecore. This model, however is not specific to the WoT and lacks important details like a clear definition/model of sensors. Additionally, the last part of the introduction showed that the community assists a shift from devices only mashups towards scenarios implying computational resources. This also motivates us to consider this type of resource. Bringing services to the WoT will finally increase its richness and usefulness. Moreover, from a user's perspective there should be no difference between a resource representing a smart-device and a resource representing some software.

5 Conclusion

Through Section 3.5 we have shown the applicability of our meta-model to real world scenarios and thus, its viability. The presented model takes into account domain specific constraints from the Web of Things community and extends this visions with the addition of service only elements. This extension becomes more and more important as recent papers shows [7]. The WoT community has steadily grown over the past few years and reaches now a state where it has to take care of formal aspects surrounding its research area. Our meta-model does not only guide the creation of models of endeavors but it also defines a vocabulary for the most important concept in the WoT. Further, it introduces the vision of an extended Web of Things, where not only smart-devices can communicate directly but they can also invoke services to improve themselves. The aim of this vision is to facilitate the integration of service elements into the WoT as it is today. Adopting the present meta-model as base, allows for a seamless integration of classic WoT components but also *marketplace* components.

References

1. Moore, G.E.: Cramming More Components onto Integrated Circuits. Electronics **38**(8) (April 1965) 114–117
2. Kortuem, G., Kawsar, F., Fitton, D., Sundramoorthy, V.: Smart objects as building blocks for the Internet of things. Internet Computing, IEEE **14**(1) (jan.-feb. 2010) 44 –51
3. Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B., Spasojevic, M.: People, places, things: web presence for the real world. Mob. Netw. Appl. **7**(5) (October 2002) 365–376
4. Guinard, D.: A Web of Things Application Architecture – Integrating the Real-World into the Web. PhD thesis, ETH Zurich, Zurich, Switzerland (August 2011)
5. Guinard, D., Mueller, M., Trifa, V.: RESTifying Real-World Systems: A Practical Case Study in RFID. In Wilde, E., Pautasso, C., eds.: REST: From Research to Practice. Springer New York (2011) 359–379
6. Gupta, V., Goldman, R., Udupi, P.: A network architecture for the Web of Things. In: Proceedings of the Second International Workshop on Web of Things. WoT '11, New York, NY, USA, ACM (2011) 3:1–3:6
7. Mayer, S., Karam, D.S.: A computational space for the web of things. In: Proceedings of the Third International Workshop on the Web of Things. WOT '12, New York, NY, USA, ACM (2012) 8:1–8:6
8. Ruppen, A., Pasquier, J., Hürlimann, T.: A RESTful architecture for integrating decomposable delayed services within the web of things. Int. J. Internet Protoc. Technol. **6**(4) (June 2011) 247–259
9. Pautasso, C.: BPMN for REST. In: 3rd International Workshop on BPMN, Luzern, Switzerland, Springer (November 2011)
10. Meyer, S., Ruppen, A.: An Approach for a Mutual Integration of the Web of Things with Business Processes. In: Proceedings of the 9th International Workshop on Enterprise & Organizational Modeling and Simulation. EOMAS '13 (2013)

11. Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodelling for software engineering. Wiley (2008)
12. Guinard, D., Mueller, M., Pasquier, J.: Giving RFID a REST: Building a Web-Enabled EPCIS. In: Proceedings of Internet of Things 2010 International Conference (IoT 2010), Tokyo, Japan (2010)
13. Haller, S.: The Things in the Internet of Things. Technical report, SAP (2010)
14. Richardson, L., Ruby, S.: RESTful web services. 1 edn. O'Reilly Media, Inc. (May 2007)
15. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. 2nd edn. Addison-Wesley Professional (2009)
16. Ruppen, A., Pasquier, J., Wagen, J.F., Wolf, B., Guye, R.: A WoT approach to eHealth: case study of a hospital laboratory alert escalation system. In: Proceedings of the Third International Workshop on the Web of Things. WOT '12, New York, NY, USA, ACM (2012) 6:1–6:6
17. Kopecky, J., Gomadam, K., Vitvar, T.: hRESTS: An HTML Microformat for Describing RESTful Web Services. Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on **1** (2008) 619–625
18. De, S., Barnaghi, P.M., Bauer, M., Meissner, S.: Service Modelling for Internet of Things. In: FedCSIS. (2011) 949–955
19. Fensel, D., Krummenacher, R., Shafiq, O., Kuehn, E., Riemer, J., Ding, Y., Draxler, B.: TSC–Triple Space Computing. e & i Elektrotechnik und Informationstechnik **124**(1) (2007) 31–38
20. Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD thesis (2000) AAI9980887.
21. Gómez-Goiri, A., Orduña, P., López-de Ipiña, D.: RESTful triple spaces of things. In: Proceedings of the Third International Workshop on the Web of Things. WOT '12, New York, NY, USA, ACM (2012) 5:1–5:6
22. Kozaczynski, W., Booch, G.: Component-based software engineering. IEEE software **15**(5) (1998) 34–36
23. Schreier, S.: Modeling RESTful applications. In: Proceedings of the Second International Workshop on RESTful Design. WS-REST '11, New York, NY, USA, ACM (2011) 15–21