Embedded Semantic Metadata to Support Device Interaction in Smart Environments

Simon Mayer
ETH Zurich
Universittstrasse 6
Zurich, 8092 Switzerland
simon.mayer@inf.ethz.ch

Gianin Basler
ETH Zurich
Universittstrasse 6
Zurich, 8092 Switzerland
baslergi@student.ethz.ch

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.
ACM 978-1-4503-2215-7/13/09.

http://dx.doi.org/10.1145/2494091.2494169

Abstract
Facilitating the interaction of human users and machines with smart devices is important to drive the successful adoption of the Internet of Things in people's homes and at their workplaces. In this poster contribution, we present an approach to support users controlling smart devices in their environment. To do this, we propose to embed semantic metadata in the representations of smart things. By means of this metadata and a semantic reasoning service, our system enables users to specify a desirable state of their smart environment and produces a machine-readable description that details which steps are necessary to reach this state, where each step corresponds to a Web request to a smart device. A client application that could, for instance, run on the user's smartphone, can distill the necessary steps required to reach the user's goal state from this description and execute them to modify the smart environment on behalf of the user.

Author Keywords
Machine-Machine Interaction, Reasoning, Smart Environments, Semantics, Web of Things.

ACM Classification Keywords
H.5.3 [Group and Organization Interfaces]: Web-based interaction.

Introduction

The *Web of Things (WoT)* is a concrete implementation of the Internet of Things (IoT) vision that focuses on establishing *application-level connectivity* between heterogeneous devices. It is based on protocols and patterns that have proven to be successful in the World Wide Web such as caching, load balancing, and searching as well as the stateless nature of the HTTP protocol. By applying these to physical devices and thus creating "smart things" that are modeled in a resource-oriented fashion and according to the Representational State Transfer (REST) constraints, it is possible to construct lightweight applications that leverage large amounts of real-time data and physical functionality. However, we expect that the abundance of such devices and of services provided by them will make it increasingly difficult for human users to find and utilize relevant services in a fast and user-friendly way, which in turn makes it hard to efficiently interact with their smart environment.

We claim that the problem of finding and using smart devices that provide relevant services for the user can be solved in WoT environments by embedding information about the capabilities of smart devices within their Web representations. Such descriptions of *what* services a smart thing provides can then be integrated with its REST program interface, to make the smart thing automatically usable by human users and machines.

In this poster abstract, we describe a system that makes use of such embedded metadata to let users specify *goals* which encode a desired state of their smart environment, e.g., to regulate the ambient temperature at their current location. The system then uses a reasoning service to determine whether these goals can be reached given the set of services available to the user. If the reasoner finds a path from the current state of the smart environment to the goal state, it produces a *proof* which details why it believes that the goal state can indeed be reached. Given this proof, our system can distill the necessary steps to reach the goal where one such step corresponds to a single Web request to a device or service, and execute them.

System Design

Our system leverages multiple services that are deployed in our research group's office space: All smart things that are connected to our local network have access to a Web-based lookup service [3] that allows to search for smart devices and also keeps track of their location. For instance, a user can query this system for all devices of type `dbpedia.org/resource/Alarm_Clock` on a specific floor of our building, or request it to deliver all devices that are located in a specific room on that floor. The infrastructure also integrates services that are not deployed on a physical device but rather can be run on any server (e.g., as a cloud service).

Embedded Semantic Metadata

The smart devices that we consider in our system feature semantic descriptions of their provided functionality and their Web API to enable automatic interaction with and between them. For instance, the lookup service provides a description that holds information about what request to send to find the URL of a service of a specific `type` at a specific `location`[1]. This metadata thus also serves to decouple the different components of our system: The functionality of the whole system is not tied to using our own lookup service. Rather, any Web service that can be sent a semantic type and a location and will return URLs

[1] The semantic categories `type` and `location` as defined by the Dublin Core Metadata Initiative (`purl.org/dc/elements/1.1`) and the W3C WGS84 Geo Positioning vocabularies, respectively.

of devices of that very type at the given location automatically integrates with our system.

To describe the capabilities of a service and link these descriptions to its program interface, we use the RESTdesc language [5] which integrates services' REST interface descriptions with metadata that is required to reason about device capabilities. The required information is encoded in the Notation3 (N3) format, which extends the RDF data model by adding assertion and logic capabilities. A concrete example of what the sematic data that is embedded in our smart things (in this case, a smart alarm clock) looks like is given here:

```
1  @prefix time: <http://[...]/time.html#>.
2  @prefix http: <http://www.w3.org/2011/http#>.
3  @prefix dbpedia: <http://dbpedia.org/resource/>.
4
5  { _:event a dbpedia:Event; time:begins ?dateTime. }
6  =>
7  {
8    _:request http:methodName "POST"; http:requestURI
         (</alarm?time=> ?dateTime); http:resp [ http:
         body ?alarm ].
9
10   ?alarm a dbpedia:Alarm; time:hasDateTime ?dateTime;
         loc:atLocation CNB/H/108.
11 }.
```

This information encodes that it is possible to obtain an instance of a `dbpedia:Alarm` that is localized at `CNB/H/108` from the response message body of an HTTP `POST` to `http://[...]/alarm?time={?dateTime}` (cf. lines 8 and 10). `?dateTime` is replaced at runtime by the variable that is linked to a `dbpedia:Event` and specifies when this event starts (cf. line 5).

To describe and classify devices in our office environment, we made use of well-known public ontologies such as the *Dublin Core Metadata Initiative* and *DBpedia*[2] whose goal it is to extract structured information from Wikipedia and make it accessible for machines. To describe HTTP requests, we used ontologies published by the World Wide Web Consortium (e.g., `w3.org/2011/http`). Devices advertise their semantic descriptions by including links to these documents in their responses to HTTP `OPTIONS` requests, as part of the HTTP `Link` entity-header[3].

Reasoning and Execution
The reasoner is a software component that generates proofs from (i) the types of available inputs, (ii) a goal and (iii) links to the semantic descriptions of currently available devices (such as the description shown above). The inputs correspond to values that the client already knows about (e.g., the starting time of an event) and the goal defines what it would like to achieve (e.g., an alarm at a specific location). As reasoning engine, we use the open-source Euler Yap Engine (EYE). Given inputs and a goal, EYE produces a proof in the N3 format from which the necessary HTTP requests to reach the goal can be extracted. We selected EYE because of its high efficiency, which allows our system to do reasoning in-between user interaction steps and without having a great impact on the system performance as a whole.

In the example above, a user would use a client application that runs on an interface device (e.g., a smartphone) to specify a goal such as "`x a dbpedia:Alarm; time:hasDateTime ?dateTime; loc:atLocation ?myLocation.`" and would furthermore enter all necessary input data regarding the event and the user's location. Given this information, the reasoner then fetches the semantic descriptions of all available services using the lookup service to produce a proof that specifies which HTTP requests are necessary to reach the goal, given the

[2] `dublincore.org` and `dbpedia.org`, respectively.

[3] tools.ietf.org/html/rfc5988

inputs. In this example, this is a single `POST` request to an alarm clock, if one such device has been found at the user's location. This request is then executed on behalf of the user by the client application, thereby modifying the smart environment to reach the desired goal. The reason for having the client execute the proof rather than letting the reasoner do this is that the alternative would raise privacy and security issues by requiring the reasoner to have full access to all devices involved in an interaction.

Related Work

The problem of specifying a *program interface* for Web services, and using such a specification to create mashups of interlinked services has already been considered, for instance, in the JOpera project [4], or in [1], where more emphasis is placed on the linking of Web resources to guide RESTful machine-to-machine interaction. An example of other approaches that tackle the problem of supporting end users in smart environments is the MIT Oxygen project, especially its *GOALS* and *MetaGlue* [2] components. Our approach is distinguished from this system as it avoids tight coupling of the involved smart things: Devices that provide appropriate semantic metadata can join our system without any reconfiguration, are immediately discovered by the reasoner and can help to work towards the user's goal by interacting with other machines. Our system is also easy to setup and maintain as all involved smart things can function by themselves, without any supporting infrastructure. Rather than tightly integrating the services provided in a smart environment, we thus use the embedded semantic metadata as a common ground for enabling automatic collaboration between smart devices.

Conclusions

We have presented a system that integrates semantic technologies and Web-enabled smart environments to facilitate the interaction between smart devices and human users. The biggest advantage of our approach is that adding a new device to an already running system is fast and straightforward. New services will seamlessly interact with other services in the system, given that their semantic descriptions are correct and sufficiently detailed. The use of semantic descriptions furthermore dramatically reduces the amount of shared information required for smart things and user interface devices to interact in smart environments. It enables applications to learn and implement all steps necessary to modify previously unknown environments on behalf of their user.

References

[1] J. Bellido, R. Alarcón, and C. Sepulveda. Web Linking-based protocols for guiding RESTful M2M interaction. In *Proc. ComposableWeb*, Paphos, Cyprus, 2011.

[2] M. Coen, B. Phillips, N. Warshawsky, L. Weisman, S. Peters, and P. Finin. Meeting the computational needs of intelligent environments: The metaglue system. In *In Proc. MANSE '99*, pages 201–212.

[3] S. Mayer, D. Guinard, and V. Trifa. Searching in a Web-based Infrastructure for Smart Things. In *Proc. IoT*, Wuxi, China, 2012.

[4] C. Pautasso. Composing RESTful services with JOpera. In A. Bergel and J. Fabry, editors, *Proc. SC*, volume 5634 of *LNCS*, pages 142–159. Springer, 2009.

[5] R. Verborgh, T. Steiner, D. Van Deursen, R. Van de Walle, and J. Gabarró Vallés. Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc. In *Proc. NWeSP 2011*, Salamanca, Spain, 2011.