Leveraging the Web for a Distributed Location-aware Infrastructure for the Real World*

Vlad Trifa and Dominique Guinard and Simon Mayer

Abstract Since GPS receivers have become a commodity anyone could access and use location information simply and freely. Such an easy access to ones' location is instrumental to development of location-aware applications. However, existing applications are static in that they do not model relations between places and mobile things. Moreover, these applications do not allow to easily map the physical location of mobile devices to virtual resources on the Internet. We attempt to bridge this gap by extending the base concepts that make up the Internet with the physical location of devices, in order to facilitate the development of Web-based location-aware applications for embedded mobile devices. In this chapter, we propose a simple infrastructure for the "Web of Things" that extends the existing Web to enable location-aware applications. The proposed solution enables a naturally hierarchic way to search for location-aware devices and the services they provide.

Vlad Trifa
MIT SENSEable City Lab, 77 Mass. Ave. MA-02139 Cambridge, USA
Institute for Pervasive Computing, ETH Zurich, Switzerland
e-mail: vlad.trifa@ieee.org

Dominique Guinard
MIT Auto-ID Labs,
Institute for Pervasive Computing, ETH Zurich
e-mail: dguinard@mit.edu

Simon Mayer
Institute for Pervasive Computing, ETH Zurich
e-mail: simon.mayer@inf.ethz.ch

* The original publication is available at www.springerlink.com published in the book: "REST: From Research to Practice", edited by E. Wilde and C. Pautasso.

1 Introduction

In the last decade, tiny computers with various sensors onboard have been increasingly installed in our buildings and cities. Connecting all these sensors to a unique infrastructure has the potential to significantly affect our daily lives by facilitating access to massive amounts of real-time data and react to various conditions rapidly. For example, a manufacturing company could monitor and detect events or anomalies in the production line rapidly, thus could react and prevent stops of production or even accidents by having an instant view of what is happening across the various locations of the company at any given time.

As the usage of such networked sensing devices will spread, efficient – yet simple – mechanisms and tools for automated data acquisition and manual interaction or control will be increasingly required. As more and more devices will need to interact and work with each other in an ad-hoc manner, an interoperable and open infrastructure for seamless integration and use of devices will become a necessity. Recent efforts in the Web of Things (WoT) [6] domain have shown that REST [4] is an appropriate architectural style for building pervasive computing applications. Various prototypes have illustrated the advantages associated with the use of Web technologies for ad-hoc interaction with devices. However, the lack of a scalable and flexible infrastructure to support and automate the search and discovery of devices based on their characteristics represents a major obstacle for building large-scale applications on top of thousands of heterogeneous and mobile sensing devices.

At any given time, any person or object has a unique location in the real world (home, office, car, etc.). In contrast, the physical location of data is irrelevant on the Web, since an efficient mechanism (universal resource identifiers, URI) is in place to access data regardless of where it is actually stored. For the Web to truly embrace the physical world, one needs to extend the classic Web model to make it easy to bind real-time contextual information to things and use this information to search things. The centralized index approach commonly used by search engines seems appropriate to store massive amounts of historical data, however when it comes to monitoring millions (or billions) of resources that will form the Web of Things, a radically different approach will be required. As more and more things will have to be monitored in real-time, a centralized repository to store and query their status as it changes would hardly scale. Present-day location-aware services such as Gowalla or Foursquare are nothing more than classical, centralized, Web applications, therefore direct, ad-hoc interaction with physical places and their services is impossible without being mediated through the remote server.

Although the cheap GPS receivers embedded in mobile phones have played a central role in the popularization of such location-aware applications, they are not useable when it comes to indoor localization. Because it does not rely on an expensive or dedicated infrastructure, Wi-Fi fingerprinting is becoming a particularly viable alternative that works at a city-scale, but also indoors. With an accuracy of

---

`http://gowalla.com/`

`http://foursquare.com/`

a few meters, room-level localization is reasonably feasible which is sufficient for most ubiquitous computing applications [1]. Even though spatial localization techniques improve over time, open and physically distributed location-aware applications are still prevented by the lack of robust and open standards for modeling and representing locations on the Web in a more flexible format than geographical coordinates [10]. Due to the lack of tools and techniques to support natively the physical location of things on the Web, discovering devices present in a certain place and interacting with them directly in an ad-hoc manner (i.e., without mediation through a centralized repository) is a complex problem which still requires custom applications. This is further aggravated by the multitude of incompatible protocols for low-power devices that coexist today. While solutions such as Bluetooth, Apple's Bonjour or Universal Plug and Play do offer powerful mechanisms for locating devices on a network, they remain overly complex and are incompatible with Web technologies and do not support the physical location of devices.

In this chapter, we describe InfraWoT, a possible solution for these problems that builds on top of state-of-the-art research in the Web of Things. We show why a RESTful architecture is an ideal solution for leveraging an existing Wi-Fi infrastructure to build a loosely-coupled infrastructure for searching and interacting with networked devices depending on their physical location. Even though the Web was designed as a hyper-linked system for multimedia documents, this chapter shows that a distributed location-aware infrastructure for embedded devices can be built solely using Web standards. In particular, we discuss how REST can be leveraged to simplify ad hoc interactions with devices by considering the spatial relations between places, devices, and people.

We extend the concept of *gateways* proposed in earlier work to connect the Web with the real world by enabling RESTful interactions with embedded devices and sensors [9]. By linking physically distributed gateways on the Web, we can form self-stabilizing hierarchical structures (trees) that can be mapped to physical locations and symbolic place concepts such as buildings, floors, rooms, etc. When new devices connect to this network through a gateway, they inherit automatically the location of the gateway they connect to, which enables physical objects to be searched, accessed, browsed, and linked together just like any other Web resource. On top of this hierarchical place model, we illustrate how Web clients can use the HTTP/URI mechanism as a lightweight and simple, yet powerful, flexible, and expressive combo to perform context-aware searches to find and use relevant objects at specific locations in real-time.

2 A Web-oriented Infrastructure for Physical Things

The success of the World Wide Web stems from its particular software architecture called Representational State Transfer (REST), which emphasizes scalability, generic interfaces, and a loose coupling between components [4]. On the Web, the primary abstraction of information and functionality are *resources* that are identified

by Uniform Resource Identifiers (URIs) and can be interacted with using the HTTP protocol. Although HTTP was designed as an application protocol with particular strengths (and weaknesses), many Web applications today reduces its role to a mere transport protocol by using only a fraction of its features. For example, Web applications that rely upon Web services based on SOAP and WSDL use only a single operation of HTTP (POST) to call API methods offered by a few URI-identified endpoints, thus hiding the actual resources being manipulated. This prevents to take full advantage of the Web architecture's features and tools (e.g., caching, load balancing, etc.), as it requires to define specific application layers for each application.

2.1 Web-enabling Things

The term *Internet of Things (IoT)* refers to networked devices with an emphasis on interoperability at the data transport layer to maximize raw performance through customized, tightly-coupled applications. More recently, the *Web of Things (WoT)* [6] vision proposed a shift towards simplified integration and programming by reusing well-known Web standards to interact with embedded devices. This way, common Web tools (e.g., browsers), interaction techniques, and languages can be directly used to program the physical world. Following the success of Web 2.0 mashups, we suggest that a similar lightweight approach for interacting with embedded devices using HTTP to manipulate URI-identified resources, significantly reduces the time required to develop applications for devices and enables the creation of *physical mashups* [5].

Another advantage of Web protocols over lower-level Internet protocols when connecting smart real-world devices to the cyberspace is that one inherits many of the mechanisms that made the Web scalable and successful like caching, load balancing, indexing, and searching as well as the stateless nature of the HTTP protocol. One can also leverage search engines to register and index a physical service (e.g., monitoring environmental sensors), by using semantic annotations to describe the functionality and interaction possibilities of each device.

Embedded devices usually have limited resources and therefore require optimized protocols to exchange data. Additionally, as HTTP or IP might not be available or appropriate for such devices, we propose to use gateways to enable Web-based interactions with low-power devices. Such a *gateway* (cf. Figure 1) is nothing more than a Web application that enables access to heterogenous devices through a simple and uniform RESTful API, thus hiding the complexity of the various protocols used by devices (such as Bluetooth or Zigbee). The gateway application is lightweight enough to run on any programmable computer with a TCP/IP connection that can run Java, as for example on open wireless routers, network-attached storage (NAS) devices, or networked media players.

Just like search engines have allowed to index and search the whole Web, we believe that the next evolution will be search of real-time data in the physical world. As demonstrated by the success of the Web, a loosely-coupled physically distributed

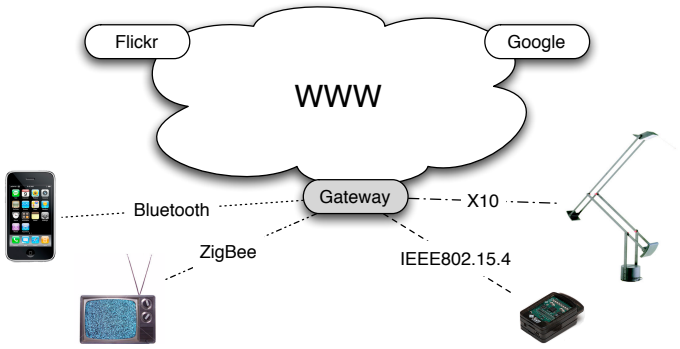

Fig. 1 A *gateway* is a Web application to bridge embedded devices to the Web by hiding the various low-power protocols used by devices behind a RESTful API.

application can scale massively and we explore how one can bind gateways and their associated devices together to form a large infrastructure that integrates all kinds of device over the Web. In particular, this infrastructure could enable to search and use devices according to their current state, location or overall context on a global scale and in real time.

2.2 Hierarchical Location Modeling

A central property of the Web is the use of hyperlinks to connect related resources on the Web, possibly using semantically annotated links (for example using *friend of a friend*, FOAF.) To create a distributed infrastructure for smart things, we propose to bind gateways together in the same manner. In previous work, we have explored how gateways can be linked to realize a distributed location-aware infrastructure for devices [8]. By assigning each gateway to a unique location and linking gateways together according to their spatial disposition, one can model the relations between places in the real world. In practice, this requires each gateway to maintain a list of links (URI) towards the gateways of (physically) adjacent places, and eventually to semantically annotate the nature of these links.

As illustrated in Figure 2, such a Web-based hierarchical model of places enables interaction with the real world with different levels of granularity (*country, region, city, street, building, floor, room, object*). Thanks to the layered system style of the REST architecture, each *node* (i.e., gateway) in the tree serves as an abstraction layer to interact with the devices and gateways contained in its subtree, thus refines its parent by offering a finer granularity to clients of the infrastructure. Such location-aware gateways are also called *location proxies*, and both terms are

www.foaf-project.org/

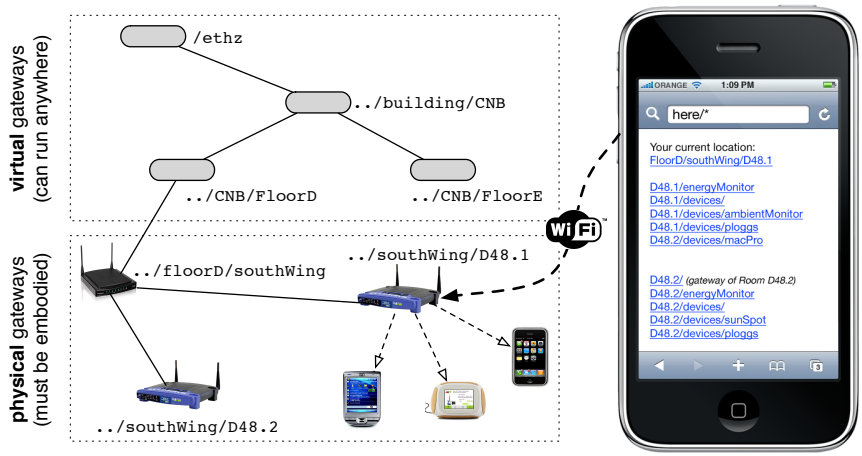

Fig. 2 Example gateway hierarchy from our building. The top gateway covers the gateways for each floor, and is composed only of virtual gateways. The `southWing` gateway runs on the router that bridges the local sub-network of that area, thus can access all terminal gateways running on computers physically located in each room. Terminal gateways have different physical interfaces to access mobile devices nearby.

used interchangeably throughout this chapter. We differentiate between two types of gateways: *virtual* and *physical*. Although identical from a software point of view, the difference lies in the fact that physical gateways (also called *terminal* gateways) must run on a computer (e.g., a wireless router, a PC, etc.) physically located in the area it maps to, and also must have different (physical) interfaces to connect with devices in that location using short-range radio protocols such as Wi-Fi, ZigBee or Bluetooth. Virtual gateways, on the other hand, do not require to be installed at the specific place that they act as a location proxy for, as they don't need to connect directly to physical devices. To give an example, the virtual gateways of the tree shown in Figure 2 can be fully distributed across servers anywhere in the world transparently as long as the logical structure of the tree is maintained.

The mapping process that assigns the logical place name (*room 44, floor D, east wing, etc.*) to gateways is done once manually by the developer at setup time. Fortunately, since gateways are not mobile and the structure of their connections is rather static, little effort is required to maintain the tree structure once designed. Terminal gateways can discover mobile devices in their surroundings and make them dynamically available as Web resources accessible over HTTP. This allows to navigate the tree by following links to surrounding gateways simply by clicking the links on a Web page or typing a URL in any Web browser.

On top of this network, one can easily build a system that supports range and lookup queries for mobile devices. Unlike most other hybrid models for spatial queries, our approach does not rely on a centralized database to store information about the system. Thanks to their RESTful interfaces, gateways are loosely-coupled

components responsible for managing the devices (and gateways) located in the area they are associated with. The more you go up in the hierarchy, the less often things are likely to change, which naturally forms an efficient load-balancing system, as users only need to access gateways located in the area of interest without soliciting the rest of the system. As the loose coupling of the location proxies furthermore allows for scalability and flexibility on the infrastructure level, this architecture is also particularly suited for ad hoc interaction with/from mobile devices that move between locations.

2.3 Localization

Given that many different localization techniques exist for different applications, the representation of the location information must be kept agnostic of the localization technique used to maximize flexibility and interoperability. Although many formats to represent outdoor locations have been developed recently, there is no standard way to represent indoor location information, and certainly none based on Web technologies. As geographic coordinates (longitude/latitude) are not practical for dealing with location concepts used in everyday life, as for example a room's number or a building wing's name, a flexible model that supports user-generated symbolic annotations of places is needed. Sharing semantics of places can be a tedious problem in case a central authority has to maintain a repository of place names, besides it would conflict with the Web's decentralized nature.

To solve this problem, we propose to use the Web itself as a lookup service to find and explore locations, as well as to obtain information about places and the devices therein. Following the idea formulated in [7], we use URIs to represent locations and their containment relations as a logical path according to the URI definition. Consequently, RESTful URIs can be created dynamically by navigating the hierarchical tree formed by the gateways. For each URI, both, machines and people, should be able to retrieve a description of the identified resource. This is essential for a shared understanding about the location identified by the URI, where machines can retrieve semantically annotated data (e.g., using RDFa or Microformats) while people can retrieve a human readable representation (HTML).

Once the gateway hierarchy is in place, the problem of determining the current location of a user on the tree still remains. In particular, when several gateways are present on the same network, how does a client know which of these is the one corresponding to its location? We call this the *bootstrap problem*, and a simple method to infer the relevant location proxy's URI based on one's current location is necessary. One possibility would be to always connect automatically to the gateway with the highest signal strength, however, in practice this turns out to be very unstable as the signal strength is subject to significant and unpredictable fluctuations.

The actual spatial localization process is not part of our project, therefore we assume an indoor localization system for finding our position at the room level. For example, we could use a system such as RedPin [2] to automatically return

the URI of the location proxy associated with the current location. The process of binding to a gateway itself should be as easy as possible, at any place where wireless connectivity is available.

Once a physical device is associated with a gateway, its location-dependent URI can be constructed using the following syntax:

```
http://host{/location}[/keyword]
```

Here, the `host` denotes the network location of the local gateway (i.e., its IP address or network name). To traverse the location structure, `/location` is used to represent a path of arbitrary length (for example `/building44/room3/`). Finally, by specifying a `keyword`, the user can search for devices and services that match the expression. With this simple syntax, URIs becomes a flexible search bar. For example, to instruct a gateway that it should return all its links (i.e., subresources) to other devices or gateways, the wildcard character "`*`" can be appended to the URI of any gateway. To find all devices tagged with the keyword `phone` located on the same floor, one can simply type the following URL in any browser:

```
http://here/floor/phone/*
```

This URI can be resolved by the access point the user is associated with by using the symbolic hostname "`here/`" - such packets could always be routed to the "nearest" location gateway which is possible because the links between gateways are tagged semantically. Subsequently, a HTML page with links to all the devices that match the query and are *under* the nearest gateway named `floor` at that time will be generated dynamically.In this setting, the same URI will yield different results depending on the node in the network that it is routed to. This allows to create fixed URIs that actually point to different resources depending on the geographic location where it is issued, which could be an interesting metaphor that many location-aware Web applications might benefit from.

3 A Distributed Modular Infrastructure for the Web of Things

A central challenge when building the Web of Things is the development of a meaningful structure on top of individual resources attached to the WoT. Because it matches the layered architecture of the Web [3], we opted for the hierarchical location model described above where each node is responsible for all devices in its proximity and every proxy at a lower hierarchical level. When following this model to map physical locations to URIs, networks of gateways automatically get arranged as a rooted tree, where the root represents the highest level of hierarchical location (for example the headquarters of an international organization). The hierarchical approach has been proposed in early research [8] and shows some benefits with respect to *load balancing* and *scalability* as users mostly access devices located in their surroundings and regarding the loose coupling between the infrastructure nodes. Our efforts towards designing and implementing such an architecture led towards the

development of the *InfraWoT* system. An important design choice for InfraWoT was that every communication between proxies does happen locally (i.e., between neighboring nodes in the tree structure). This helps to scale the infrastructure, as each gateway only requires knowledge about its direct neighbors and thus can remain ignorant of the remaining hierarchy.

Selecting information on the hierarchical location as the main structural descriptor has immediate implications on several components of the infrastructure, for instance on the service responsible for querying within the InfraWoT tree structure or on the module in charge of maintaining the correct infrastructure internally (i.e., deciding which gateway to choose as parent and which to accept as children).

3.1 Modules Overview

As flexibility is a key requirement when implementing such an infrastructure, we decided to create location proxies that would be able to be reconfigured without requiring a restart. To achieve this level of flexibility, we have chosen the *OSGi framework* as it supports component-based development and future component-level upgrades which fosters "hot-pluggability" with other software developed for the Web of Things.

The data format used for internal information transfer is the *JavaScript Object Notation (JSON)* interchange format that provides very lightweight and easy-to-use encoding and decoding of data. Another advantage of using JSON as data format comes from the human-readable structure of this format, which greatly simplifies the debugging of software infrastructures using logs and/or live monitoring of the message streams exchanged between gateways.

The InfraWoT software consists of several modules that can interact with each other via OSGi-based messages. Each module is responsible for a specific task and implements an interface that gives access to a limited set of framework-wide functions. Figure 3 presents an overview of the different modules in each InfraWoT node.

- **Infrastructure Service Module** This module maintains the correct tree structure with respect to the hierarchical locations of other proxies within its scope. As such, it takes care of child/parent registration and generates maintenance traffic between directly connected proxies (i.e., between parents and their children).
- **Discovery Service Module** This component handles the discovery of resources, in particular the retrieval of information on resources that are to be integrated into the infrastructure and the mapping of this data to internal representations. Through this process, newly discovered resources get attached to the tree hierarchy via an InfraWoT node and thus can benefit from the services offered by the infrastructure.

`http://www.osgi.org`

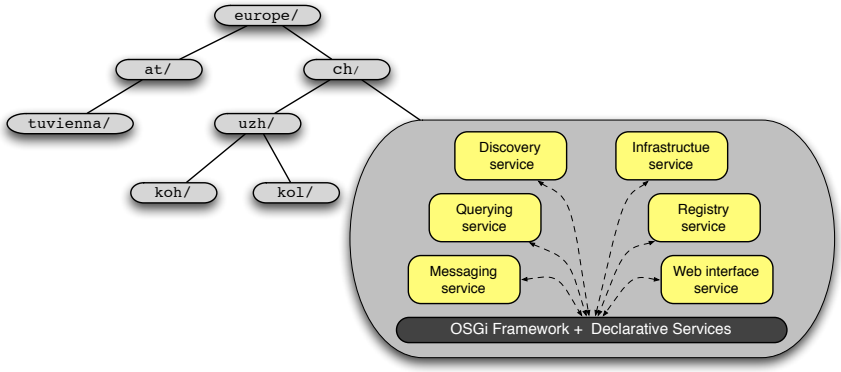

Fig. 3 Modules running in each InfraWoT node that interact via OSGi Declarative Services.

- **Registry Service Module** This component manages data about attached resources (both locally connected devices and neighbor gateways) and stores this information into a local (typically embedded) database.
- **Messaging Service Module** This module offers a transparent interface to set up a messaging (i.e., publish/subscribe) system between client applications, gateways and physical devices attached to the Web of Things.
- **Querying Service Module** This module is responsible for handling incoming queries. It retrieves local resources that correspond to the query and forwards the query to suitable sub- or super-nodes.
- **Web Interface Module** This module provides a Web interface that allows to access the various functions offered by the gateway, either via a RESTful API or via an actual Web-based user interface accessible from any browser. The Web server is built upon Restlet and offers various device- and gateway-specific functions.

3.2 Device and Resource Discovery Service

When a new device is connected to a network, an automated mechanism to detect the new device and to extract information about it and on possible interactions with the device is necessary. Many discovery protocols exist (WS-Discovery, Bonjour, etc.), however, most of them are overly complex and require an implementation of the complete discovery protocol on each device. The solution that we propose fully leverages REST to minimize the infrastructure changes required to use InfraWoT in a large-scale scenario. Furthermore, devices do not need to implement any specific discovery protocol, but rather just have to provide semantic information about

`http://www.restlet.org/`

themselves in their root document. In this section, we will describe the process of attaching a new resource (i.e., a networked device featuring a RESTful interface) to the InfraWoT system.

Device Discovery

The first step of the discovery process deals with finding new WoT devices that are connected to a network. Here, we do only assume Ethernet/WiFi-enabled devices as, for other protocols, a gateway is necessary.

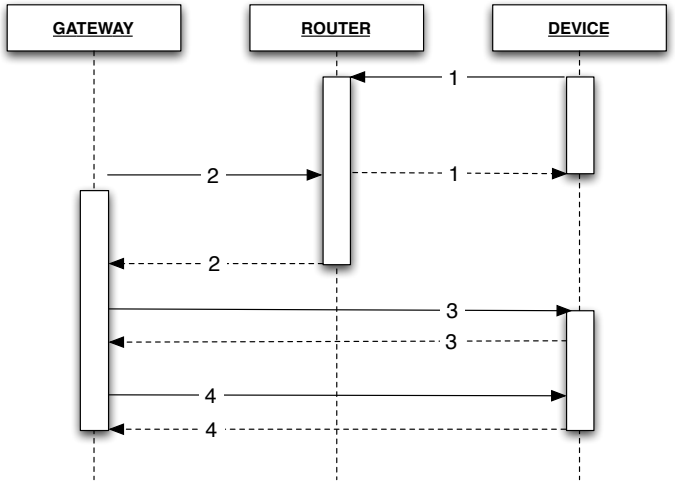

Fig. 4 Sequence diagram of the RESTful discovery process of devices. 1. Device connects to LAN/WiFi and gets an IP address from the router using DHCP. 2. The gateway monitors the router's DHCP table. 3. For each new device found, the gateway retrieves the root device page (by default a HTTP server running on port 80) and parses it to find information about the device. 4. The gateway retrieves the semantic description of the device.

Most existing discovery solutions rely on devices multicasting UDP messages over the network. However, as such messages are not part of HTTP, they can often be blocked by firewalls. We therefore propose a REST-based protocol to perform network device discovery. We assume that in each network, the router is always knowledgeable of the connected network devices (usually a table of automatically assigned IP addresses), and as such can provide all required discovery information. To access this information, our solution uses *OpenWrt*, a is widely used open source Linux distribution available as firmware for many modern network routers. Its user

`http://openwrt.org`

interface - *LuCi* - exposes some of its libraries and functions to external applications through a JSON-RPC API.

To retrieve the list of all connected devices, an HTTP request is sent to the router (Listing 1):

```
Method: POST
URL: http://router/cgi-bin/luci/rpc/sys?auth=
    EBAE1814FA625E73CA0514004428D64A
Content-type: application/json
Content: {"jsonrpc": "2.0", "method": "net.arptable", "id":
    1}
```

Listing 1 Example of an authenticated POST command to retrieve the list of devices connected to the router from LuCi.

This request will return the list of devices connected to the router by calling the RPC method `net.arptable`. Listing 2 shows a typical message returned by this call:

```
{"id":1,"jsonrpc":"2.0","result":[
  {"Flags":"0x2","HW type":"0x1","Device":"br-lan","Mask
      ":"*","HW address":"00:E0:4C:45:57:EF","IP address
      ":"192.168.1.114"},
  {"Flags":"0x2","HW type":"0x1","Device":"br-lan","Mask
      ":"*","HW address":"00:1C:B3:25:F6:9B","IP address
      ":"192.168.1.149"},
  {"Flags":"0x2","HW type":"0x1","Device":"eth0.1","Mask
      ":"*","HW address":"00:0D:66:22:38:01","IP address
      ":"89.211.57.1"}]}
```

Listing 2 Example device listing response from LuCi.

The response includes a list of the IP addresses of all the physical devices connected to the router together with additional useful information. Once a list of the IP addresses of new devices that have just connected is retrieved by a proxy, the root page of each device is parsed by the Discovery Service using the procedure described in the next section (by default, the root page should be located at `http://[IP_address]:80/`).

Resource Discovery

Once a new device has been connected to the network, the second discovery step (resource discovery) is carried out to retrieve various information about the device (functions/services, description, etc) and make this information available within InfraWoT. In case it cannot be triggered automatically by the devices discovery process describe in previous section (in case the router doesn't offer the list of its routing table through a Web API), one needs to manually `POST` the URI of the device

`http://luci.subsignal.org`

The current version of the LuCi is not RESTful, but as it is an open source project, the RESTful equivalent of this procedure can be easily implemented.

root page to the `/resources` endpoint of a gateway. Such requests are unpacked by the Web Interface module and the payload is relayed to the Discovery Service Bundle.

InfraWoT provides a discovery service for Web of Things resources that is based on multiple semantic identification strategies. When a new resource is being discovered by InfraWoT, it is analyzed and mapped to an internal resource representation according to semantic markup that the resource may provide. To extract this data, InfraWoT tries to interpret any accessible representation of the resource using a number of different discovery strategies. Depending on the specific strategy, the string representing the resource is interpreted differently, for instance as a URL or as a JSON-encoded resource description. Additionally, the InfraWoT infrastructure takes into account the location information that may be provided by a discovered resource and takes care of registering that resource with the best-suited location proxy by forwarding the registration to a parent- or child-node, respectively. The different strategies have been implemented as a *Strategy*-pattern and can easily be extended, for example by implementing parsers for RDFa- or XML-based resource descriptors.

In the current version of InfraWoT, two strategies have been implemented. In the first one, InfraWoT searches the HTML resource representation found at the device URI for Microformats. Microformats provide a simple way to add semantics to Web resources. There is not one single Microformat, but rather a number of them, each one for a particular domain; a geo and adr microformat for describing places or an hProduct and hReview microformat for describing products and what people think about them. Each Microformat undergoes a standardization process that ensures its content to be widely understood and used, if accepted. More concretely, InfraWoT understands a compound of several, optional, Microformats that can be used to better describe devices. This helps for devices to be searched by humans using traditional or dedicated search engines (e.g., Google or Yahoo which are both supporting Microformats), but it also helps them being "discovered" and understood by InfraWoT in order to automatically index and use them. Currently, InfraWoT supports, but does not require, five Microformats; *hProduct* is used to describe the device itself (brand, name, picture, etc.). *hReview* reflects the quality of service or experience users or applications had with the device, *hCard* and *Geo* specify the location context of the device (address, region, country, latitude, longitude, etc.).

Finally, *hRESTS* is used to provide additional information about the REST services that a device offers directly embedded in its HTML representation. An example of the hRESTS markup to describe, for example, the *Light Sensor* resource of a sensor node is shown in Listing 3. It is worth noting that most of this information could be inferred by crawling the HTML representation of resources of a (truly) RESTful device and using the HTTP `OPTIONS` method. However, having this information directly embedded in the human representation of a device presents some advantages such as minimizing the HTTP calls on the device or being able to render device user interfaces in a special way, highlighting the offered services for human

users. As an example, Google and Yahoo use a special HTML rendering for search results containing pages that embed Microformats such as *hReview* and *hCards*.

```
<span class="hrests">
  <span class="service">
    <span class="operation">
      The
      <span class="label">Light Value</span>
      operation returning the
      <span class="output">current light value</span>
      can be invoked using a
      <span class="method">GET</span>
      at
      <span class="address">../{device}/sensors/light</span>
    </span>
  </span>
</span>
```

Listing 3 Microformats annotations used to describe a device and its operations, in this case a photosensor of a sensor node.

The second type of discovery strategy that is currently supported by InfraWoT is based on interpreting the resource representation as a JSON object according to a pre-defined, fixed, schema. While this is not realistic on an Web-wide scale, it can be used in controlled environments (e.g., in an intranet or behind proxies such as gateways) as it is much more efficient than the Microformats-based discovery because there is no need to parse the entire device root page to find the embedded semantic annotations.

Thanks to the modular architecture of InfraWoT proxies, additional strategies can be *injected* in the Discovery Service at runtime by POSTing them to the `/strategies` endpoint of the proxy. The discovery mechanism of InfraWoT is very permissive as the minimal information necessary about a resource is the URI of that resource. If a resource provides a unique identifier within its representation, that data is incorporated as the resource's internal Unique Universal Identifier (UUID). Else, the proxy registering the object generates a new ID for unique identification of that resource. Every piece of additional information that a resource offers (e.g., using Microformats) is used to extend the resource's internal representation and thereby enables more services for that resource, most prominently advanced support for querying and location-aware registration.

3.3 Querying Service

Querying for resources within the scope of specific locations (such as *"find all printers in this room"*) is a central feature of any infrastructure for smart devices. InfraWoT enables such queries using various parameters such as the name of resources,

---

`http://microformats.org/2010/07/08/microformats-org-at-5-hcards-rich-snippets`

their description, or the RESTful operations and parameters they accept. Additionally, InfraWoT defines several query types that encapsulate scoping information (i.e. *where* to search for resources). The handling of a search request is thus a two-step procedure that consists of first routing a query to the most appropriate gateway (e.g., the location proxy responsible for a specific building or a certain room) and then triggering it and returning the discovered resources.

A client may submit a query by sending an HTTP `POST` request to the `/query` endpoint of a proxy that contains a description of the query either as a JSON encoded string or using a collection of form parameters. Internally, queries are represented as JSON-serializable Java objects that contain (as mandatory parameters) an ID, the URL of the proxy that initiated the query and their type. Additionally, a query may contain an arbitrary amount of (optional) parameters that are added to the JSON representation when serialized. Such an open design facilitates upgrade and maintenance of InfraWoT (for instance, one could have queries carry piggyback structural information). HTTP responses to client queries can be delivered in multiple different formats, depending on the HTTP `Accept`-header specified in the request (usually JSON/XML in queries from another node/application, HTML in queries from a browser).

In principle, proxies should enable querying for all parameters that occur in the internal representation of resources. Our implementation is currently limited to those parameters that are most valuable for clients of the infrastructure – a client may search for resources using the following query types:

- **Keyword Queries** Keyword queries have become - thanks to the popularity of Web search engines - the most intuitive query format for many users. *Structured* queries (i.e., classical database queries) are quite complex for humans, who would rather provide textual information about the object in demand, and let the querying mechanism carry out the interpretation of this data. InfraWoT supports simple keyword-based querying by matching the provided keywords with the multiple properties and descriptions of every device in the database.
- **UUID Queries** Particularly useful for machine-machine interaction, using the unique identifier of a device is needed when an application wants to use the infrastructure to interact with the same specific device over and over again. To humans, UUID queries are only of limited use because of the numeric format of device IDs.
- **Name Queries** These queries enable clients to search for resources by their name and thus represent the human-useable version of UUID Queries.
- **REST Service Queries** Matching resources according to the REST services they offer is an essential enabler for machine-machine interaction. As the devices we have enabled for the Web of Things implement the *hRESTS* Microformat, their HTML representation contains human-readable descriptions of their capabilities – every resource that offers services specifies its functionality with the associated *label*, HTTP *method*, *input*, *output* and *address* information, where the *input* and *output* specifiers provide a for machines to index keywords about the services of resources.

3.4 Infrastructure Service

The Infrastructure Service is used to initialize the tree structure at startup time and ensures that the correct structure is maintained during operation. In particular, this service allows the overall structure to recover from node failures and eventually re-establish the initial tree configuration (self-stabilization). After the initial setup, all gateways initialize their Infrastructure Service bundles which start the registration process with their assigned parents by sending an HTTP `POST` request that includes their own URI. Every gateway that receives such a request forwards the received URI to the Discovery Service which adds the respective gateway as a new child node.

Furthermore, the Infrastructure Service is responsible for the process of attaching new sub-resources (i.e., other proxies or devices) that are found by the Discovery Service or registered manually. Any resource that is encountered and analyzed by the Discovery Service is passed to the Infrastructure Service which uses the resource's hierarchical location information to determine whether to attach it to the current proxy or to send it to a more appropriate gateway. In the latter case, the infrastructure takes care of routing that resource to the proxy whose hierarchical location corresponds best to the resource's (cf. Figure 5). If a registering resource does not provide location information within its Web representation, the Discovery Service automatically assigns the location of the proxy itself.

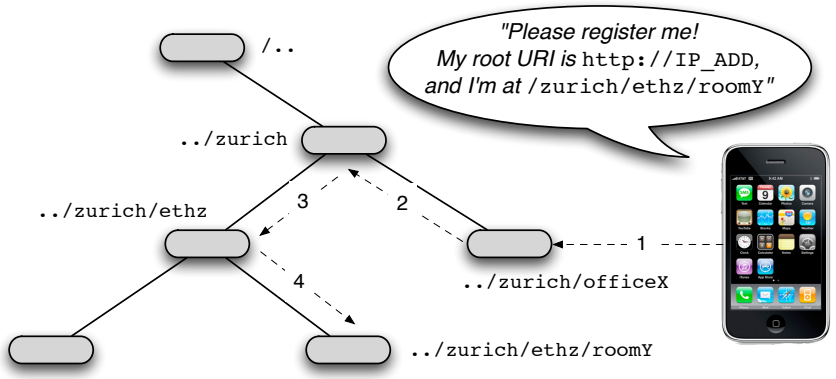

Fig. 5 Infrastructure-assisted discovery: any device can POST its root URI (that contains semantic information about itself) to any node in InfraWoT. If the optional location is known, the registering in routed to the node corresponding to the location specified.

Finally, it acts as *garbage collector* by regularly contacting the sub-resources and removing them from InfraWoT when they become unavailable. The Infrastructure Service starts two threads that regularly contact the parent node, all registered chil-

dren nodes, and all attached resources. If the connection to any of these resources is lost, the corresponding entity gets black-listed and will be removed if contact cannot be re-established after a timeout period.

3.5 Web Interface Service

The Web Interface Service enables to access the infrastructure and the various resources connected to InfraWoT using only RESTful requests. In particular, the Web Interface Service enables the RESTful configuration of InfraWoT location proxies. We now briefly describe the individual endpoints of the InfraWoT Web interface and their functionalities:

The *root* of any gateway ("`/`") provides general information on the current proxy (name, hierarchical location, connected sub-nodes, attached resources, etc.). From the root, one can access four different sub-resources (in addition to `/query` described in Section 3.3):

The `/locations` resource represents a list of all attached location proxies. Child nodes may send HTTP `POST` requests to this address to be registered by the proxy. The HTML representation of this resource can be used to navigate (browse) the infrastructure. The individual gateways registered to any node are represented as child resources of the `/locations` resource, which can also be used to delete child nodes. For instance, to remove the gateway with UUID *ID32*, an HTTP `DELETE` request should be sent to the resource `/locations/ID32`.

The `/resources` resource represents a list of all sub-resources attached to the current gateway. Similar to the `/locations` resource, Web of Things resources may send HTTP `POST` requests to this resource to be registered by the gateway. Likewise, these resources are represented as child resources of the `/resources` resource and can be interacted with via requests to their respective endpoint within the local gateway.

The `/infrastructure` is mainly used internally by the InfraWoT software to send and receive maintenance information. One of its sub-resources, though, plays an important part in the fully Web-based configuration system of InfraWoT that enables clients to configure a proxy by sending HTTP `POST` requests to its configuration interface at `/infrastructure/configuration`. When a client `POST`s a string of data to this endpoint, the proxy relays that data to the Discovery Service to retrieve the resource encoded in the transmission and applies that information to its own representation. Although the currently preferred way to configure a gateway is to `POST` the desired configuration as a JSON-encoded resource, a gateway can be configured using any representation that is supported by the Discovery Service.

The `/messaging` resource and its sub-resources handle all interaction related to the InfraWoT Messaging Service, i.e. the creation, updating and deletion of information on the messaging interface between the gateways, client applications and physical devices.

Finally, the `/strategies` resource allows to inject additional discovery strategies at runtime (cf. 3.2).

4 Discussion

In this chapter, we introduced InfraWoT, a flexible and scalable infrastructure for a new generation of Web applications that integrate real-time information from the physical world. As an extension of previous work with Web-enabled devices and gateways [6, 9], InfraWoT fosters the rapid development of scalable distributed applications that incorporate data and/or functionality from heterogeneous resources on the Web of Things.

Using RESTful patterns to connect individual gateways to form a structured network that models the spatial hierarchy between places, the real-time context of Web resources (e.g., their current location) can be integrated into the Web fabric in a natural and efficient manner that allows for Web-based context-aware discovery, search and use of devices and resources. A possible scenario for this would be searching for restaurants in the vicinity according to their real-time situation (crowded, noise, etc.), by directly querying the local network infrastructure without having to fetch specific centralized Web sites.

Such a scenario could be easily implemented using InfraWoT with little or no infrastructural changes. Any existing Wi-Fi network in place would be sufficient to run InfraWoT as the discovery/query procedures are fully based on Web standards. For example, the resource discovery process we proposed can be used directly as long as a gateway and a WoT device are on the same network, by POSTing manually the root URI of the device to the gateway. This procedure could be performed automatically if routers could expose network-level information through RESTful APIs. A world where every device, gateway, or router in a network could host a local Web server offering a JSON-based API for applications and a HTML-based user interface for human users, is technically feasible today (an increasing number of routers, printers, and consumer appliances on the market today have embedded Web servers, or at least a Wi-Fi/Ethernet interface).

By describing how various functions useful for building more interactive pervasive applications can be implemented using REST, we have shown the practical advantages and flexibility offered by REST when applied to physical computing. Thanks to the layered system offered by REST, which bounds the overall system complexity and promotes the loose coupling between components, different parts of the network can be implemented independently according to the specific requirements of different applications. Using a uniform, RESTful interface for every Web of Things resource would facilitate ad hoc interaction with/between them. This way network-level information (e.g. routing tables or network load) and real-time data from the physical world (through sensors, etc) could be seamlessly integrated into Web applications, therefore opening a whole new range of design possibilities to make the Web *more physical* and *more real-time*.

Security and privacy issues have not been addressed in this chapter, however we are investigating the use of HTTPS and OAuth to enable authenticated and secure communication between mobile clients and gateways. A detailed performance and scalability analysis of the whole system will be required in order to understand how to develop much larger systems.

This chapter offers a window on what the future Web might look like, and we hope to inspire Web developers to think about new possibilities that arise when combining a truly location-aware infrastructure with the Web. The work presented here shall not be taken as a finite solution, but a mere prototypical draft to foster the exploration of a future Web of Things. Much applied research and prototypes will be required before device-oriented standards for the Web become widely adopted, however, we hope our initial results and positive experiences with REST on embedded devices will stimulate further efforts to construct the Web of Things.

References

1. Abowd, G.D., Mynatt, E.D.: Charting past, present, and future research in ubiquitous computing. ACM Trans. Comput.-Hum. Interact. **7**(1), 29–58 (2000)
2. Bolliger, P.: Redpin - adaptive, zero-configuration indoor localization through user collaboration. Workshop on Mobile Entity Localization and Tracking in GPS-less Environment Computing and Communication Systems (MELT), San Francisco (2008)
3. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
4. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Internet Techn. **2**(2), 115–150 (2002)
5. Guinard, D., Trifa, V.: 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 WWW (International World Wide Web Conferences), Madrid, Spain (2009)
6. Guinard, D., Trifa, V., Wilde, E.: A resource oriented architecture for the web of things. In: Proceedings of the IEEE International Conference on the Internet of Things (IoT 2010). Tokyo, Japan (2010)
7. Jiang, C., Steenkiste, P.: A hybrid location model with a computable location identifier for ubiquitous computing. In: Proceedings of the 4th international conference on Ubiquitous Computing, pp. 246–263. Springer-Verlag, Göteborg, Sweden (2002). URL http://portal.acm.org/citation.cfm?id=741480
8. Trifa, V., Guinard, D., Bolliger, P., Wieland, S.: Design of a Web-based Distributed Location-Aware Infrastructure for Mobile Devices. In: Proc. of the First IEEE International Workshop on the Web of Things (WOT2010), pp. 714–719. Mannheim, Germany (2010)
9. Trifa, V., Wieland, S., Guinard, D., Bohnert, T.M.: Design and implementation of a gateway for web-based interaction and management of embedded devices. In: Proceedings of the 2nd International Workshop on Sensor Network Engineering (IWSNE'09). Marina del Rey, CA, USA (2009)
10. Wilde, E., Kofahl, M.: The locative Web. In: Proceedings of the First International Workshop on Location and the Web, pp. 1–8. ACM, Beijing, China (2008). DOI 10.1145/1367798.1367800