Semantic Interoperability Requires Self-describing Interaction Models

HATEOAS for the Internet of Things

Matthias Kovatsch
Siemens AG / ETH Zurich
kovatsch@inf.ethz.ch

Yassin N. Hassan
ETH Zurich
yhassan@student.ethz.ch

Klaus Hartke
Universität Bremen TZI
hartke@tzi.org

Abstract—Most efforts for semantic interoperability in the *Internet of Things* (IoT) focus on the information model. This is indeed a crucial aspect to make use of the data produced by IoT devices. To enable robust machine-to-machine communication that can handle change in a global system, however, semantic interoperability also requires a machine-understandable interaction model. In the World Wide Web, the *Hypermedia as the Engine of Application State* (HATEOAS) principle of REST is such a mechanism that provides functional descriptions for the APIs. The interaction is modelled in atomic operations (request-response exchanges resulting from links or forms) that have semantic annotations to describe the function (link and form relation types). We argue for a combination of information model semantics (e.g., RDF) with interaction model semantics (i.e., hypermedia controls) that make API operation explicit, and hence machine-understandable.

Introduction

The Internet of Things (IoT) is expected to connect numerous digital services with physical objects. Many of the devices will be autonomous, that is, they perform machine-to-machine communication without human interaction. For this to be possible, the complete interaction must be machine-understandable. This includes two parts:

- An information model that describes the exchanged information, and
- an interaction model that describes the possible interactions with a service.

Various standardization groups have defined data formats such as IPSO Smart Objects [IPSO] or the ZigBee Cluster Library [ZCL], which also have a conceptual model for the data. Semantic models such as the Resource Description Framework (RDF) [RDF] could be used to build an information model across the different consortia, as they usually share a common meta model. The advantage of semantic information models is that machines can interpret meaning from the data itself without the need to know the meta model. Furthermore, it allows for the interlinking of data models, so that a model from one application domain can be integrated with another [LD].

Orthogonal to that, interaction models tell a machine how an API is used and can drive the interaction during runtime. Usually, this part is hard-coded in machine clients: A developer uses a textual description of the API to implement the necessary protocol steps. Here, more formal description such as RAML [RAML] can help to generate the necessary stubs. This approach is not a problem when it is cheap to upgrade all clients to the new API version, which is usually the case for smartphone apps that

talk to cloud back-ends of the same vendor. However, when billions of small, independent nodes provide services to billions of clients, the cost of breaking the API is very high, as a developer has to intervene and upgrade all clients to the new service API.

When a machine client can discover the possible interactions with a service, it could adapt to changes by itself. The REST architectural style [REST] provides such a mechanism for the interaction model: Hypermedia As The Engine Of Application State (HATEOAS). Here, the server provides in-band descriptions of the API through the publication of links and forms. Only the atomic interaction steps (following links and submitting forms) have to be shared a priori. Each interaction step corresponds to a request-response exchange. This mechanism lets machines not only handle change over time, but also enables interoperability between changing APIs from different manufacturers.

Hypermedia Controls

Hypermedia is a well known concept in the Web. There, people find information by following links and interact with Web services by using forms. The idea of self-describing interaction models is to apply this concept to machine-to-machine communication. The crucial point here is that the machines need to understand what the links and forms mean. This is in contrast to the Web, where a human user interprets the links and forms in their context and knows which link to follow or how to fill out a form.

Link relation types [RFC5988] is a concept that attaches machine-understandable meaning to links. Link relation types are used in the Web when the Web browser needs to discover by itself without human intervention, for example, the stylesheet of a Web page. Link relation types are also used in microformats to attach machine-readable meaning to links, and in the Atom Syndication Format [RFC4287]. By defining IoT-specific link relation types, it is possible to drive interactions through links instead of hardcoding URIs into the client, thus making the system flexible enough for later changes.

While link relation types are quite established in the Web, forms do not enjoy a standardised mechanism for attaching meaning yet. However, it is easily imaginable to have something akin to a "form relation type" that serves for forms the same way as link relation types for links. Yet a form enables a richer interaction than a link, as the client can include data when submitting the form. So, in addition to the form relation type, a machine-understandable form also needs to include a machine-understandable description of the accepted data, e.g., as a set of machine-understandable input fields similar to microformat properties.

Hypermedia Client

While the design of the right hypermedia formats and vocabulary for the IoT still requires research effort, the embedding of hypermedia controls on the server side is relatively straight-forward. The consumption on the client side, however, is challenging. We believe that the lack of proper abstractions to define the behavior and goals of a hypermedia-driven machine client has led to the pre-dominance of quasi RESTful applications, where the consumption of a REST API is hardcoded in the client. Thus, most APIs also miss the hypermedia controls and no IoT-related vocabulary for relation types could emerge.

Ideally, developers would program the behavior of a hypermedia-driven machine client by describing

- the intended Web resource based on its context, independent from the current network addresses and server structures, and
- the intended resource state based on an information model that can be applied to the representation formats and forms.

To specify the context of a resource, one or more link relation types need to be given. This usually starts from the entry URI, from where the client performs an incremental discovery of related resources.

Prototype Implementation

We implemented this abstraction using Futures and provide a Java-based prototype together with JavaScript wrapper module for scripting[1]. The description of an intended resource looks as follows:

```
client = new HypermediaClient("coap://home.local"); // entry point
resource = client.follow("my-lightbulb-tag") // give link relation types
                 .follow("lighting-state"); // returns a Future
representation = resource.get(); // lazy evaluation of the Future
```

When the developer knows what representation formats to expect [CA], the program API (and hence the IDE autocomplete feature) can directly offer format-specific methods to consume data or submit forms:

```
temp = representation.getColorTemp(); // reads color temperature
newRepresentation.setHSV(120, 1, 1); // sets color to green
representation.formEdit(newRepresentation) // returns a Future
              .get(); // lazy eval. of the Future (i.e., submit form)
```

Often, finding the correct context requires active exploration of multiple links (e.g., to find a lightbulb at a specific location). Furthermore, it can happen that the client arrives in an unknown context, where none of the links provide a known relation. In both cases, a crawler that provides specific strategies (e.g., crawl for a hierarchical location identifier) can help:

```
thing = client.links().use(new ThingCrawler())
                      .findLocation("/CH/ETH/CAB/51")
                      .findFirstWith("lighting-state");
```

Optimizations

The incremental discovery in hypermedia-driven applications might appear inefficient due to the high number of roundtrips, especially for constrained IoT devices. However, there are several optimizations that can be applied and usually directly result from the REST architectural style. All optimizations are implemented in our Hypermedia Client.

Caching

Representations can be cached directly on the client or on less constrained intermediaries. Thus, not all interactions require communication, as requests can be directly answered from the most local cache. For this to work, it is important that the server provides deliberate caching control information (e.g., Max-Age option).

Bookmarks

Bookmarking is an explicit caching strategy controlled by the client. Once the Web resource of interest is found, the client can store the effective URI and always start the interaction from there (e.g., read a

[1] GitHub repo URI t.b.d. (will be publicly available on GitHub after clean-up)

sensor value or set a parameter). In addition to the URI for the bookmark, it is useful to also store a recipe how the resource was discovered. In case the resource becomes unavailable, a client can then efficiently rediscover a resource that fulfills the same purpose (e.g., when the light bulb was replaced or when the API changed).

Reduced Representations

Similar to bookmarks, clients could cache the discovered interaction model and request an optimized representation format that omits the hypermedia controls. For instance, when having discovered a temperature sensor, the client is only interested in the sensor readings, but does not require links for further interaction. This can be achieved through reduced representations or a format that only carries data without controls.

Related Work

RESTdesc [REDE] is a community effort to provide functional descriptions for REST APIs (i.e., an interaction model). Given a goal, this approach uses first-order logic to produce an execution plan from inference rules over semantic facts (“triples”). Each rule is annotated with with a REST request that transforms the set of given preconditions into a set of postconditions. This way, the proof for achieving the goal also contains an ordered list of requests, which represents the execution plan. The drawbacks of RESTdesc are the performance of semantic reasoners when the number of services and goal complexity grows, the corruption of the whole system when a single functional description is faulty, and a cumbersome syntax to formulate complex goals.

CoRE Link Format [RFC6690] and CoRE Interfaces [CI] provide attributes that are related to link and form relation types. The resource type (rt) is a tag to identify a resource. Opposed to link relation types, it is an absolute definition, while link relation types are relative and can be combined to incrementally specify the context of a resource. The interface type (if) is close to a form relation type: It specifies the possible interactions with the resource, while the content type (ct) attribute tells the client which representation formats are supported. However, the method and required input fields are implicit and have to be taken from the specification.

The W3C WoT Interest Group has designed a Thing Description (TD) [TD] that provides metadata and simple interaction patterns (Properties, Actions, and Events). Rooted in RDF by using JSON-LD [JSONLD], the metadata is machine-understandable and the model can easily be extended with domain-specific semantic descriptions, which can even be translated automatically through ontology mapping. The interaction model of TD, however, is still implicit and the construction of requests is hardcoded in clients. Thus, the simple interaction pattern cannot evolve over time without breaking deployed applications. Yet by including hypermedia controls to describe the patterns, the interaction model would become explicit in machine-understandable form, allowing for a complete solution.

Conclusion

A global information model alone does not solve semantic interoperability. Besides correctly interpreting data, machines must also be able to understand the API semantics to automatically adapt to change. Hypermedia controls provide this for the World Wide Web; however, only for human users. Machine clients are still programmed with an implicit interaction model tied to a specific API and API version. As reason for that, we identify the lack of a proper programming abstraction for hypermedia-driven machine clients.

We experiment with a prototypical hypermedia client that uses Futures to provide developers with a way to describe Web resources based on their context. This way, environments and server structures can change over time without breaking the running applications, since the client is able to rediscover its resources of interest on a replacement device or in a new API version. This shows how hypermedia controls can enable the IoT to handle change.

It has proven useful to rely on a closed set of vocabulary for individual IoT applications. The vocabulary used can be grouped and identified by Internet Media Types. By following the recommendation of [CA], which is that an application needs to list the Internet Media Types used, a machine client can be sure that it will be able to understand all the vocabulary needed to fulfill its goal. Otherwise, constrained devices would need to be prepared to dynamically load new vocabularies and perform semantic reasoning steps.

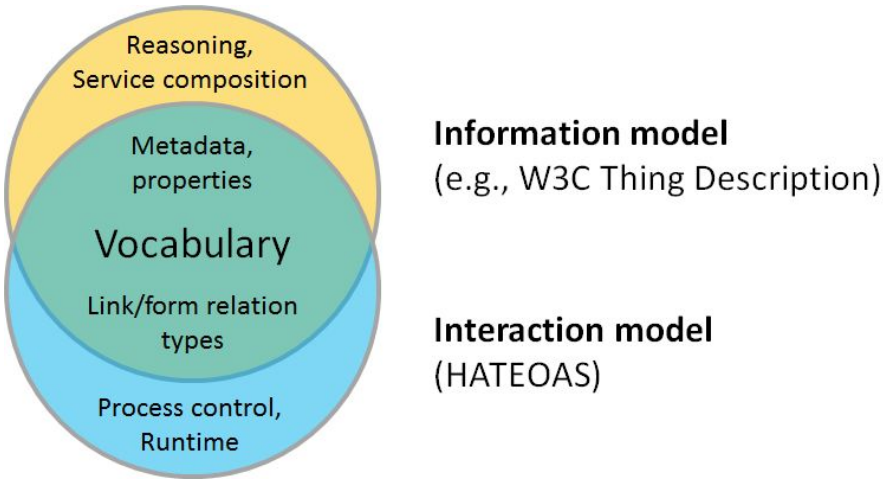

Figure 1: The symbiosis of information and interaction model

Ultimately, we envision the combination of a self-describing interaction model using hypermedia controls with a powerful information model as depicted in Figure 1. Semantic reasoning over the information model on the one hand can be used for the evaluation of metadata, automatic service composition, and validation of data flows. On the other hand, hypermedia controls ensure interoperability at runtime. Client and server can evolve over time and new components can be introduced without breaking the application. This symbiosis would provide the advantages of both worlds and allow for semantic interoperability in the IoT.

References

[CA] K. Hartke. “CoRE Application Descriptions”. I-D.hartke-core-apps-03, 2016.

[CI] Z. Shelby, M. Vial, M. Koster. “Reusable Interface Definitions for Constrained RESTful Environments” I-D.ietf-core-interfaces-04, 2015.

[IPSO] IPSO SmartObject Guideline. Technical Guideline (Version 1.0), IPSO Alliance, 2014.

[JSONLD] M. Sporny et al. “JSON-LD 1.0 - A JSON-based serialization for Linked Data”. W3C Working Draft, 2013.

[LD] T. Bizer, T. Heath, T. Berners-Lee. “Linked data - The Story So Far”. Semantic Services, Interoperability and Web Applications: Emerging Concepts, 2009, S. 205-227.

[RAML] “RAML 1.0”. Technical specification (RC), 2016.

[RDF] G. Klyne, J. Carroll, B. McBride. “RDF 1.1 Concepts and Abstract Syntax”. W3C Recommendation, 2014.

[REST] R. T. Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. PhD thesis, University of California, Irvine, 2000.

[REDE] R. Verborgh, V. Haerinck, T. Steiner, D. Van Deursen, S. Van Hoecke, J. De Roo, R. Van de Walle, J. G. Vallés. “Functional Composition of Sensor Web APIs”. In Proc. SSN, Boston, USA, 2012.

[RFC4287] M. Nottingham, R. Sayre. “The Atom Syndication Format”. RFC 4287, 2005.

[RFC5988] M. Nottingham. “Web Linking”. RFC 5988, 2010.

[RFC6690] Z. Shelby. “Constrained RESTful Environments (CoRE) Link Format”. RFC 6690, 2012.

[TD] D. Peintner, M. Kovatsch. “WoT Current Practices”. Unofficial Draft, 2016.

[ZCL] ZigBee Alliance. “ZigBee Cluster Library”. ZigBee Document 075123r04ZB, 2012.