A Semantic-aware Framework for Service Definition and Discovery in the Internet of Things Using CoAP

Farzad Khodadadi¹, Richard O. Sinnott¹

¹Melbourne eResearch Group, School of Computing and Information Systems
The University of Melbourne, Parkville 3010, Australia

Abstract

The future of machine-to-machine (M2M) communications relies on reusability, scalability, and interoperability of services. Internet of Things (IoT) aims to provide an environment where smart devices can easily expose their services, while providing accurate service discovery functionality for clients to consume them. Semantic-first approaches and service annotation techniques have already been studied in the service definition context and most of them leverage WSDL or RESTful services. However, IoT environments, due to their heterogeneous nature and often constrained resources, demand a more flexible and scalable approach. In this paper, semantic annotation of services is supported through ontologies defined in API definition languages such as Swagger and RAML. We leverage the CoAP protocol and linked-data serialization format (JSON-LD) to represent services, entities, and properties in a semantic-aware framework. This enables intelligent discovery of services. A case study on reducing energy consumption in data centers is used to demonstrate the effectiveness of our proposed framework.

© 2017 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the Conference Program Chairs.

Keywords: Internet of things; service discovery; linked data; semantics; CoAP protocol

1. Introduction

The term Internet of Things (IoT), first introduced by Ashton in 1999, envisions the idea of interconnecting smart devices and physical objects, referred to as Things, to the worldwide Internet network, while exposing their features and services to authorized entities in a secure manner. This context-awareness can make our surrounding objects smart and allow them to share their status or react to external stimuli. Being context-aware and able to understand and exchange domain-specific information are fundamental requirements of successful cross-domain applications.

The surge in data generated by devices, humans, autonomous software systems, and many other resources has introduced serious challenges in terms of data acquisition, storage, and knowledge extraction. This is exacerbated...
when dealing with inter-disciplinary domains. The IoT which promises to enhance interoperability and connectivity among the vast network of smart devices and their controlling authorities can add to the aforementioned data deluge problem due to their proliferation. Since heterogeneity is one of the core characteristics of IoT, services need to be decoupled and self-informative.

Service-oriented architectures (SOA) and methodologies have been widely adopted and studied in distributed systems, well before the emergence of IoT. However, due to huge number of entities and large diverse service pool in IoT, the trend has shifted towards using more lightweight services. Traditional SOAP-based services have been gradually replaced by RESTful services and APIs are now the new players in this field. APIs are easier to define, invoke, share, and monitor compared to other service definition methods. Enterprises can commercialize their APIs more effectively, since more tools and documentation is available for end-user consumers. Generally, IoT environments span across multiple domains and to make interoperability between different domains functional, one beneficial approach is using semantic technologies. To achieve this goal, we need to combine the power of semantic-aware service discovery with modern and standard service definition methods that take into account the special requirements of IoT such as limited and scattered resources. In this paper, we propose a system that facilitates service definition by leveraging API definition languages and implements an efficient discovery component by semantically tagging services. The approach utilises both CoAP and HTTP requests for service discovery and semantics storage.

Sharing service domain knowledge by leveraging semantic technologies is a common approach for implementing semantic interoperability across various domains. Accordingly, an integrated service infrastructure and discovery framework that includes the required components for storing, indexing, and querying shared semantic data can be a good candid solution. However, current solutions merely support minimal representation of resources and provide basic discovery of resources and properties based on text matching. In many cases, considerable human effort is required to design, deploy, and link modern IoT and Semantic Web of Things (SWoT) applications. The framework developed in this paper overcomes many of these issues.

The rest of the paper is organized as follows: Section II provides a literature review of relevant works and terminologies. Discussions of Web of Things and how it can be implemented using RESTful services, Linked Data and the CoAP protocol are present in Section III. Section IV contains our proposed system architecture along with details of its major components. Section V presents the case study and is followed by the conclusion and future works section.

2. Related Work

Software typically consists of different components and each component consumes and exposes a small number of well-defined services. When services are well documented and defined, system integration can be much easier and consequently, application and user interactions with the system become smoother and more transparent. The effect and importance of services has been the causing factor behind emergence and evolution of Service Oriented Architectures (SoA) alongside service definition languages such as WSDL for SOAP and recently, Web APIs for documenting RESTful services.

IoT is based on communication of sensors and machines, hence a vast body of literature can be found on approaches that aim to make the sensory layer smart and context-aware and benefit from the Semantic Web. SSN-XG developed by World Wide Web Consortium (W3C) and OntoSensor are two prominent reference ontologies for data annotation in Semantic Sensor Networks (SSN). In 6, the authors propose an application that is capable of extracting and measuring functional, non-functional, and other properties of sensors. They show how the OWL-defined ontologies for these properties and other available services can form a composition system to meet diverse user demands. Linked Open Data (LOD) is another solution considered in various researches. LOD provides guidelines for merging and linking newly created semantic data with existing ontologies. Gramegna et al. propose a SSN-based framework that implements a backwards compatible CoAP extension to discover and match not only semantic-enriched sensors, but also annotated events.

Generally, there are two approaches for semantically annotating and representing services. The first approach—known as semantic first—emphasizes the separation of semantic service representation from the underlying technical description (for example WSDL for SOAP or API description documents for REST). OWL-S and WSMO follow this methodology and include a grounding procedure that links semantic and technical discrisions. On the other hand, WSDL-S and other similar approaches embed semantic annotations into technical descriptions directly. In
the authors propose a lightweight ontology for semantic annotation (SAWSDL—Semantic Annotations for WSDL and XML Schema) of service descriptions and introduce hRESTS—a microformat RDFa-like method—for including semantic information directly into HTML.

Desai et al. discuss the benefits of having a Semantic Gateway as a Service (SGS) model to bridge raw sensor information with knowledge-centric applications. The proposed approach supports multi-protocol proxies with their associated interfaces that are linked to a semantic annotation service. Although CoAP and MQTT are named as potential proxies with SSN as the domain-specific ontology supported, the framework merely focuses on service annotation. In our framework, service discovery using text-based and semantic-based approaches are supported, while a CoAP proxy smartly replaces the heavy-weight HTTP protocol for the end devices that support it.

3. Everything as a Service Paradigm

Heterogeneous environments such as IoT are comprised of various entities that are often geographically distributed. In a typical architecture, networks of sensors, actuators, and smart devices typically comprise the bottom layer, while gateways, mobile networks, and telecommunication facilities act as middleware and link the former layer to the cloud. Although human interaction with machines has been studied in other domains, most IoT-focused solutions either completely neglect the role of humans, or consider them as mere controllers or end users. On the contrary, we envision humans as an important part of IoT systems that can offer various services according to their role and capabilities. With this in mind, we propose a framework that models all available resources as services, whether they are sensors, edge devices, cloud resources, or conventional services. In addition, humans are also modeled as services, enabling them to interact with services exposed by other entities, thus creating an IoT mashup. Recent advances in defining and controlling hardware configurations through software layers, e.g. software defined networks (SDN), have boosted the practicality of moving towards Everything as a Service (EaaS) paradigm.

In the Web of Things (WoT), machines cannot easily interpret high-level application semantics due to the lack of control over hypermedia. We address this shortcoming by incorporating Linked Data best practices in our service and resource definition model. Moreover, since all involved entities are defined as services, we need a uniform interface for service description. As discussed earlier, WSDL was the first attempt to standardize WS-* family of services, however due to its heavy-weight nature and reliance on XML as the only protocol for serialization, it has been gradually replaced by RESTful service representation. In REST, resource state can be accurately represented and exchanged through standard URIs and HTTP protocol methods, which results in loosely coupled services.

More recently, Web APIs have gained momentum as a game-changing technology built on the grounds of RESTful web services, but with enhanced features. Swagger, now also called Open API Specification, was introduced in 2010 and is one of the pioneers in the API documentation field. After forming the Open API Initiative under the Linux Foundation in 2015, Swagger has focused on standardizing its services and boosting its presence in industry. RESTful API Modeling Language (RAML) is another major API definition language that aims to make API-first design simple and API consumption frictionless. Both of these technologies are language agnostic and human-and-machine readable, which means they are good candidates for agile software development environments where services should be delivered to customers quickly. A comparison between REST, Swagger, and RAML is presented in Table 1, explaining features of each method.

To address the aforementioned challenges and to leverage state-of-the-art technologies, we opt the Thing Description Document (TDD) specification introduced in and use JSON-LD serialization format and external ontologies that are defined as new contexts. A TDD consists of two major parts: resource properties and the services offered by each entity.

**Resource Properties:** The first part of any TDD is dedicated to describe different properties of the associated entity. A user-chosen name for faster and easier discovery with information about the entity’s unique URIs, name, and type are mandatory parts of every TDD, but any other required data field can also be defined as long as it conforms to JSON validation rules.

---

1. http://swagger.io
Table 1: Comparison of RESTful web services with Web APIs

<table>
<thead>
<tr>
<th>Feature</th>
<th>REST</th>
<th>Swagger</th>
<th>RAML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports standard HTTP methods (POST, GET, etc)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Built-in or third-party provided API description editors</td>
<td>No</td>
<td>Yes, through Swagger Editor</td>
<td>Yes, through API Designer</td>
</tr>
<tr>
<td>Supports stub and server code generation</td>
<td>No</td>
<td>Yes, through Swagger code-gen</td>
<td>Yes, through third party libraries and tools</td>
</tr>
<tr>
<td>Support for automatic unit testing against implemented code</td>
<td>No</td>
<td>Yes, through third-party frameworks</td>
<td>Yes, limited support compared to Swagger</td>
</tr>
<tr>
<td>Visual frontend for interacting with APIs</td>
<td>No</td>
<td>Yes, Swagger UI</td>
<td>Yes, API Workbench</td>
</tr>
<tr>
<td>Supported formats for describing services</td>
<td>N/A</td>
<td>JSON, YAML</td>
<td>YAML</td>
</tr>
</tbody>
</table>
grounding mechanism to generate semantic annotations according to the predefined ontology and then stores the results as RDF triples. The semantic grounding process is discussed in more details later.

When a request arrives to search for resource(s) that match certain criteria, in the first instance the query parameters are extracted. Initially, a request is submitted to the keyword-matching search engine to quickly lookup the results. Matches found in this step are then substituted in the SPARQL query that runs against deployed RDF database. The semantic-based discovery module is implemented as a RESTful web service and can be called from a CoAP endpoint in addition to normal HTTP method calls. A great advantage of having a discovery module as a web service is that the results from multiple calls to this service can be easily compared together. This allows for chaining relevant resources that match certain criteria and hence define goals than can be achieved using a composition of service calls and/or access to resource properties. More details on the practicality of service composition using this approach can be found in \(^{15}\).

Storing and retrieving semantic data is not as fast as fetching data from a relational database and the performance degrades further when the data size—depth of the semantic graph—grows and this is especially problematic if a reasoning engine is used to infer logic from statements. To support large scale IoT applications with numerous resources, we avoid using a reasoner in our implemented semantic engine, even though it is an option that can be added at a future time.

Syntax matching helps to improve the overall search performance since it stores indexes like a search engine and later, when a query is submitted, supplied keyword(s) can be retrieved much faster, compared to the semantic discovery approach where the whole semantic graph needs to be traversed. In our framework, resource properties, service names, and all descriptions and metadata are indexed. Thus when searching for a keyword, if a match is found, the TDD of the corresponding resource will be returned.

4.2. CoAP Integration

Constrained Application Protocol (CoAP) is a lightweight UDP-based protocol developed with constrained devices in mind. It helps reduce the overheads of using normal HTTP requests and responses. One benefit of using CoAP is that IoT devices will consume less power and achieve better performance. To support CoAP-enabled devices in our proposed service discovery system, a proxy server is introduced to handle communications that use this protocol. The proxy server knows which device is CoAP-ready and chooses the right communication protocol according to each device’s capability.

When a request arrives to call a specific service of a resource, the "uris" property of the resource’s TDD is examined and if it has a reference to a CoAP endpoint, the HTTP methods implemented in CoAP will be used to make the associated service calls. This approach benefits constrained devices by saving the total bandwidth required to send service call requests and receive responses. Furthermore, the CoAP proxy service at the server supports resource
discovery, which means instead of using the heavier HTTP protocol to discover other resources and services, semantic-based discovery and keyword matching can be performed directly using the more lightweight CoAP protocol.

For simplicity and enhanced integrity, the framework automatically handles TDDs that include at least a single CoAP URI in their specification and registers these resources with the Resource Directory (RD). Whenever a list of TDDs is required, a POST request to the "./well-known/core" URI returns all registered TDDs.

4.3. Semantic Grounding

Technically, semantic grounding involves relating the representational model to its corresponding semantic model, or to put it another way, it is about linking syntactic and semantic representations together. Annotation methods like RDFa include semantic information in the actual model description, e.g. HTML, while approaches like ours adopt a bottom-up preference. There is a need to have a procedure that links these two approaches.

We use JSON-LD as the serialization format to define properties of resources. JSON-LD allows external vocabularies and ontologies to be defined using its "context" value. In addition to this feature that results in a convenient way of annotating resource properties and including custom ontologies, our proposed framework supports automatic annotation of API descriptions that are specified in each TDD. APIs that follow Swagger or RAML specifications are analyzed using their corresponding parser with their semantic model generated with respect to our developed ontology and finally stored in the triple store.

Each API description document consists of several sections. The most important ones are: path of resources, consume/produce data type(s) for each resource, response type and body structure of response messages, HTTP methods defined for each resource, security schemas if any, and metadata such as API version, description, and examples. The framework supports Swagger and RAML as two leading API definition languages, and since they both have all the aforementioned sections but adopt different notations, we need to define a common ontology to represent vocabularies of both languages. Figure 2 demonstrates a minimal ontology for modeling both RAML and Swagger API descriptions.

We adopt the ontology presented in17 and extend it to include vocabularies and ontologies that are essential for defining RAML-based API descriptions. Due to space limitations the complete ontology is not presented here. It can be accessed from the Github repository3.

5. CASE STUDY: Saving Energy in Data Centers

To understand the benefit of the framework presented, we consider a case study focused on energy saving in data centers (DC). Energy consumption of a DC depends on numerous factors. For example its planned capacity, the heating and cooling infrastructure required to maintain the correct temperature according to the DC’s climate zone, high voltage power suppliers, UPS facilities to handle electrical interruption, and many more. All these factors, directly or indirectly, relate to the DC’s total energy consumption and define how green a DC is compared to standard measurements. Nowadays, in addition to scientific workflows and complex commercial applications, even services such as email are being hosted in the cloud (due to its scalability and efficient pay-as-you-go pricing model), hence energy efficiency of DCs has become a hot topic and a lot of research is going on in this regard.

According to our assumption, every element of an IoT environment has a description document that points to its properties, as well as exposed services, using the previously discussed design patterns. In this context consider a cloud provider that operates and maintains several DCs and in each one the following sensors, services and features are available:

- A thermostat that is capable of measuring and reporting the temperature of each rack at specified intervals;
- A smart multimeter that calculates and reports total voltage and electrical current consumption of each rack;
- Heating and cooling facilities that are able to adjust the temperature of the server room with regards to the specified target temperature;

---

3 https://github.com/farzado/serviceOntology
• Each server can be uniquely identified according to its rack and position number and offers services that report its current load (specially CPU and RAM utilization data) at specified time intervals.

An environment that satisfies the above prerequisites can be easily constructed considering today’s available technologies and when all resources and services comply to our proposed model, we can leverage our semantic discovery and flow composition procedures to combine these features and move towards the ultimate goal, which is optimizing the total energy consumption of the DC. A typical cloud provider can use the services offered by smart thermostats, multimeters, and load level of each server at any time to get an overview of energy consumption within a given DC. An example policy for adjusting the server room temperature according to actual server loads is presented below.

If the load of the server $s_j$ is denoted by $l_j$ at any given time $t_i$, then for any window size $k$, the total load of the servers $s_i$ to $s_j$ in the $[t_i, t_{i+k}]$ time frame can be calculated according to Formula (1).

$$L(k) = \sum_{i=1}^{i+k} \sum_{j=1}^{J} l_{ij}$$ (1)

Next, we use the weighted average function defined in Formula (2) to calculate the average load of servers, denoted by $EL$ at any time period, according to past observations and the current load of the servers. $\alpha$ is the weight parameter and can be set according to the DC’s historical load variance.

$$EL(k, k-1) = \alpha \ast L(k) + (1 - \alpha) \ast L(k - 1)$$ (2)

The power consumption of each rack can be calculated using the formula below and measurements of voltage and current from the smart multimeter. For simplicity, we use the mean of power values as an indicator of power consumption during any window size.

$$Power(t_i) = Voltage(t_i) \ast Current(t_i)$$

With the calculated values from Formulas 1 and 2 and current temperature recorded for each rack during a specified time frame, we can calculate the correlation between power consumption of each rack and its associated temperature. This value alongside other technical parameters of a DC can be used to set the server room temperature to an optimal value, and since it has been shown that cooling and heating infrastructures within DCs are the main power consumers, this optimization can potentially lead to a substantial reduction in power bill.

Note that the above scenario is only possible if all resources, from sensors to actual physical hosts, expose their properties and servers in such a way that cloud providers can use our framework’s resource discovery and service composition features to find their desired services. With the inherent potential of the flexible resource discovery module to leverage semantics in searching for sensors that are in this case, in a specific location and return their measurement in a specific format, service integration and composition can be more effectively applied.

In addition to resource discovery, the framework utilizes a CoAP proxy that smartly identifies CoAP-enabled devices and prioritizes the usage of CoAP instead of the HTTP protocol to reduce the overall network bandwidth and energy consumption. For instance, when 1000 new entities are registered and the TDD for each one contains 3 services and 2 properties, running a SPARQL query over the CoAP endpoint reduces the bandwidth by 15%. When the number of TDDs increases or TDDs have more services and/or properties, the response message will be larger, hence using HTTP methods over CoAP results in less bandwidth and power consumption. Efficient usage of the UDP protocol instead of TCP and the light-weight structure of the CoAP protocol are the main reasons for this efficiency. In the framework, Californium (CF)\textsuperscript{18} — a Java-based CoAP implementation — is used to implement the CoAP proxy.

The proposed framework is capable of reducing overall energy consumption of data centers using its two key modules: the discovery module that provides an interface to query available resources and services, hence facilitating the communication and linkage between entities; and the CoAP integration module which autonomously detects CoAP-enabled devices and leverages HTTP methods over CoAP to call their services. The discovery service can also be invoked using the CoAP protocol, thus providing a mechanism for querying constrained devices with less overhead.
6. Conclusions and Future Work

In this paper, we presented an IoT-focused framework for defining and discovering services and resources. In particular, we adopted the notation for resource definition from WoT and extended it by supporting semantic and syntax-based discovery functionalities. Furthermore, inclusion of the latest technologies for service definition such as Web APIs and addition of a CoAP proxy layer supports flexible and scalable resource discovery not available in other approaches. We also demonstrated a case study on energy savings within IoT-enhanced cloud data centers that defined resources and services in a discoverable and searchable fashion. The framework supports the composition of these elements while is oriented for scalability.

One improvement over the currently proposed framework is to use stream-based query languages such as C-SPARQL and Continuous Query Execution over Linked Stream (COELS) instead of SPARQL. Considering the heterogeneity and diversity of IoT environments, more often data is generated as continuous streams. As such, it is essential to use stream-based query languages over the resultant RDF streams. However, currently available languages suffer from scalability and performance degradation issues. Furthermore, the resource discovery component of our framework can be enhanced by inclusion of semantic reasoners, however, due to the heterogeneity and scale of IoT environments, this solution might not scale and perform well in applications with large numbers of entities and ontologies.

7. Acknowledgement

This research is supported by an Australian Government Research Training Program (RTP) scholarship.

References