Simurgh: A Framework for Effective Discovery, Programming, and Integration of Services Exposed in IoT

Farzad Khodadadi, Amir Vahid Dastjerdi, Rajkumar Buyya
Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: {fkhodadadi@student., amir.vahid@, rbuyya@}unimelb.edu.au

***Abstract*—While Internet of Things has emerged as a great opportunity for industrial investigations and similarly pursued by research communities, current architectures proposed for creation of IoT environments lack support for efficient and standard way of discovery, composition services, and their integration in scalable manner. We propose Simurgh, a framework to leverage modern state-of-the-art techniques and standards to define, discover and compose "things" and their corresponding services. Our approach allows for efficient discovery of IoT devices and their exposed services, while also considers humans as main players. This new approach facilitates communication between involved entities by forming a ubiquitous environment of IoT elements, described using standard human-and-machine-readable files, which can easily find each other and call advertised IoT services using standard RESTful web APIs. Furthermore, by chaining IoT service calls together to form flows and then combining and orchestrating these flows, end-users can achieve their desired functionality without having to worry about programming skills.**

I. Introduction

Number of smart devices is estimated to outnumber living human beings in 2017 [1]. Considering the fact that many of these devices use Internet as their backbone for communication, Internet of Things has emerged as a new paradigm aiming at providing solutions for integration, communication, data consumption and analysis of these devices. The Internet of Things (aka IoT) is the term used for drawing a circle around sensors, actuators, smart devices and in general, every object capable of communicating via Internet to form a collection of identifiable smart entities [2].

IoT promises a new world of connected devices and humans in which managing infrastructures and cities are less cumbersome, health services are conveniently accessible, disaster recovery is more efficient, and in summary a superior life quality is achieved.

Although new smart devices and their connectivity present potential for innovation, they are not playing the main role in the market. The real demand is derived from applications consuming data that IoT devices generate. The main issue here is having a standard, comprehensive and yet simple framework to make use of IoT devices. Moreover, another key challenge in a distributed, heterogeneous, and mobile environment such as IoT is properly handling "things" discovery and services they provide, yielding to facile integration with other networks and services, thus resulting in a ubiquitous and more coherent environment.

In today's market, we have to look beyond existing customers to unlock IoT services for wider set of user applications. We address aforementioned adoptability issues by introducing Simurgh[1], a framework that not only precisely defines "things', humans, and their specific properties, but also facilitates service definition and IoT service composition in form of flows. Furthermore, with burst of resources made available by different organizations, companies, and websites, there has been long effort in research community to standardize the way these resources can be exposed and securely accessed by authorized end users. Service-oriented architectures, including SOAP-based approaches and RESTFul methods are two examples of these efforts. Recently, large attention is absorbed by the concept of web APIs and how they can be defined, shared, and accessed easily. Hence, Simurgh will leverage RESTful API Modeling Language (RAML) [3], a standard API definition language, to extend this notion to objects defined in IoT environment, thus making resource and service exposition more simple and practical. By using this framework, large number of entities can be defined and leveraged, utilizing definition files created by vendors and maintained in an easy-to-use repository.

The rest of this paper is organized as follows. Section II presents related works and positions them in relation to the proposed framework. Section III explains the architecture and the components of Simurgh framework. Details about Thing Description Document and discovery component is given in IV and V, respectively. Section VI introduces the envisioned case study for framework utilization along with implementation

[1]Simurgh is a mythical bird in Iranian mythology. In the story, a band of thirty birds is searching for the Simurgh to be their king. They are looking for a superior creature as each of them represent a human imperfection. Finally, once they arrive at the place of the Simurgh, what they can observe is a lake in which they see their own reflection. This resembles a well-connected and integrated IoT environment.

978-1-4799-8325-4/15/$31.00

details and how the flows can be created using our proposed framework. Conclusions and future works are discussed in Section VII.

II. Related Work

In terms of communication protocols, Constrained Application Protocol (CoAP) promises to bring RESTful web services to constrained and embedded devices. As it is being standardized, researches [4]–[6] have been conducted to investigate its effectiveness in IoT domain. However, since the concept is new and has not been adopted by industry leading companies, practical usage of proposed solutions remains unclear.

Mayer et al. [7] proposed a web-based infrastructure for smart "things" to facilitate their integration and look-up process. Different markup languages such as Microdata and Microformats are used in their work for describing smart devices, but most of these languages are now considered obsolete.

In [8] authors argue about using Web Service Description Language (WSDL) and Microformat-based markup languages to implement a discovery and selection framework for WS-* and RESTful services. Although their solution uses machine-readable files for resources API exposition, they do not support location-based discovery of devices, which is included in our proposed framework. In a similar way, the author in [9] proposes a central repository that registers Web-based things by assigning each one a unique tag.

Li et al. [10] and Klauck et.al [11] suggest a SOA-based service discovery framework in IoT which leverages Domain Name System (DNS) to map available objects to domain names by encoding devices information and then discover them through XML files representing different service categories. However, considering huge number of devices that will be involved in a typical IoT environment, this mechanism doesn't provide a scalable and robust solution.

Briefly, other works [12], [13], whether opting syntax-based discovery method or semantic-based one, do not consider user requirements in developing IoT applications, although, efficient discovery of resources is an essential part of this process. We try to address this issue in our proposed framework by embedding discovery component as a service in user-creatable flows.

III. Proposed Architecture

Our proposed architecture offers an integrated solution that uniquely applies state-of-the-art technologies to simplify discovery, accessing, and programming of functionalities and services offered in IoT environments. An important goal of this architecture is to simplify management of IoT services, things, humans (experts and volunteers) who are involved in processes designed in smart environments enabled by IoT technologies.

The main objective is to achieve easy process design, service reusability, openness and security at the same time. In the architecture, this is achieved mainly through exposing IoT services as APIs and compose them via creating API flows to achieve more complex functionalities which can be wrapped

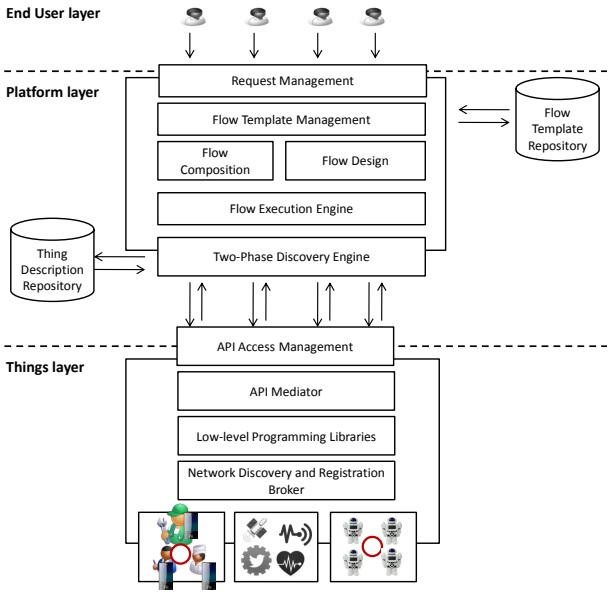

Fig. 1: Simurgh framework overview

to flow templates. The process of designing and composing flows can be carried out by an expert and then used as normal user application. This approach, considerably reduces the overhead of accessing and programming IoT services for majority of users and increases their adoptability. The proposed architecture is depicted in Figure 1 and its main components are described below:

Things Layer

- **Network Discovery and Registration Broker:** This component handles incoming connection requests from devices that want to join the domain. It also keeps a repository of entities with their assigned unique identifiers.
- **Low-Level Programming Libraries:** It provides device-specific interfaces to access functionalities of devices made by different vendors.
- **API Mediator:** API(Application Programming Interface) is an interface that allows a programmer to expose functionality of his application in forms of callable functions with precisely defined input and output types. Generally, APIs are made of libraries that include specifications for data structures, procedures, variables and object classes. A class of APIs which we focus here is known as "Web APIs" and has these main features:
  - is invoked over Internet
  - almost always uses HTTP (or HTTPS) as the main communications protocol
  - uses XML, JSON, Object types, or plain text to represent a response
  - often uses either HTTP query parameters or XML documents to represent a request

  Web is switching to Web API/REST, because older service-oriented methods are more complicated to develop and require much more resources (RAM, band-

width, Computation resources), as a result of all request-response data conversions that should take place. Furthermore, the standard form of data exchange in SOAP based web services is XML, which can efficiently be replaced by more lightweight formats such as JSON.
Building APIs facilitates and speeds the way organizations can share their services with users, helping them attract more people while focusing on the functionality of their products rather than on presentation. Considering multi-tenancy security features of modern Web APIs such as OAuth, APIs are capable of boosting an organization's service exposition and commercialization, while also providing better service monitoring tools than previous service-oriented approaches.
API Mediator provides a wrapper around low-level interfaces of devices that do not have RESTful API. When dealing with humans, the mediator shows what devices and operations are linked to a person and how he/she can be reached by communication means such as email and SMS. All provided API Mediators and their corresponding low-level libraries are kept in a central repository, which enables easy search and access method for connecting to a device, calling its services, and updating core functionalities. This component can also take the responsibility of converting already defined services that do not use RAML format for service definition.
- **API Access Management:** Once IoT Services are exposed beyond the firewall, we require a way to handle identity and access management from a centralized management component. This component utilizes state-of-the-art protocols such as OAuth and OpenID Connect and provides secure and auditable access to IoT services for end users, regardless of the device, application or location.

Platform Layer

- **Thing Description Repository:** It contains description of things and humans and services offered by them. The repository is constantly updated by the Network Discovery and Registration Broker and API Mediator component.
- **Two-Phase Discovery Engine:** This component allows for discovery of "things" that not only have capability of fulfilling the user request, but are also equipped with APIs that can be utilized in a flow to satisfy user requirements.
- **Flow Design:** It provides a user interface that allows users to discover things and their APIs and call found APIs. Additionally, this component is responsible for considering human in loop interactions with smart devices (i.e. finding and calling necessary communication APIs such as Email, SMS, etc).
- **Flow Composition:** This component is capable of combining two or more flows to build a new flow that delivers new functionalities.
- **Flow Execution Engine:** Once a user asks for a flow execution, this engine for a given number of times, or for a provided time period, provisions required resources, configures them and then execute all necessary APIs in order to fulfil the request. This component is also responsible for provisioning resources for testing the flows once they are being designed.
- **Flow Template Management and Repository:** Flow Template Repository contains pre-designed flows that are likely to be reused. Template management tasks including access control, persistence, test, and update is carried out by Flow Template Management component periodically.
- **Request Management:** This component matches the user request to flow templates. If no match is found, the request will be forwarded to Flow Composition module to check whether any composition flow can match the requirement. Finally, if no flow is found, user will be presented with a Flow Design interface to build the required flow.

IV. Things Description Document and API Definition

To describe building blocks of an IoT environment including devices, sensors, and humans, a file should be created for each of these entities, detailing its specific features and callable services. For this purpose, a lightweight human-and-machine readable file format is required to simplify writing descriptions without adding extra overhead. However, due to the lack of standardization in this area, we have developed our own meta-model called Thing Description Document (TDD) in the form of a JSON file. A TDD file consists of two major parts: 1) IoT entity properties 2) Services offered by each entity.

1) Entity Properties: The first part of any TDD as shown in Listing 1, lines 2-21, is dedicated to describe different properties of its related entity. A user-chosen name for faster and easier discovery with information about last modification date and entity's location are mandatory parts of every TDD, but any other required data field can also be defined in this section, as long as it conforms to JSON validation rules. In lines 32-38, it is declared that this sensor is being operated by a human. Later, when we explain the discovery phase, managing human interactions in IoT and how to discover and leverage them is detailed.

2) Entity Services: After describing properties, there is a dedicated section to define APIs that are available by the entity being described. Lines 22-31 in Listing 1 demonstrate how this objective is achieved. The API definition files can have RAML or Swagger [14] format type and each one can be parsed by an appropriate parser and called using their specific client tools. These TDDs with API definition files will be kept and indexed by TDD Repository component to be later searched by Two-phase discovery component.

Listing 2 depicts a minimal and simple RAML file used for defining and describing a single service offered by a typical person. When a connection to the endpoint hosting this web service is initiated, the service implementation is supposed to respond with the email address of this person by accepting an identifier that has numeric type. As it is evident, using an API

definition language provides opportunities for developers to define and expose their implemented services easier and also augments the clarity of final result, compared to approaches using SOAP and WSDL.

A. Motivation Behind Using RAML to Expose IoT Services

Several open-source API definition languages and tool-sets are currently available, with the two most renowned being RAML [3] and Swagger [14]. We use RAML in our implementation because of having better integration with our flow composition engine. Although both formats can be used when referring to API definition section in TDD, when RAML is used, after the discovery phase and given the discovered TDD, the services can be easily called using RAML client tools that are available as part of API flow composition tools such as Anypoint Studio [15]. This makes API discovery, flow execution, debugging, testing, and deploying more convenient. Furthermore, RAML allows for easy integration of security mechanisms like OAuth and provides option for including schemas that define the actual query parameter types each service accepts.

Listing 1: Typical description document for a thermal sensor

```
{
  "name": "Thermal Sensor TDD",
  "description": "This is the description document for a
       sample Thermal sensor",
  "last-modified": "2014-12-15",
  "tags": [
    "thermal",
    "Celsius",
    "sensor"
  ],
  "id":"192.168.1.3:683C.B35A.17DB",
  "location":
  {
    "Room":712,
    "Building":"Doug McDonalds",
    "Number":10,
    "Street":"Swanston",
    "City":"Parkville",
    "State":"Victoria",
    "Country":"Melbourne",
    "Postal code":3000
  },
  "api": [
        {
          "type": "Raml",
          "name": "thermal-tdd.raml"
        },
        {
          "type": "Swagger",
          "name": "thermal-tdd.swagger"
        }
      ],
"operated-by": [
        {
          "Full Name": "Farzad Khodadadi",
          "email": "info@khodadadi.com",
          "sms": "xxxxxxxxx"
        }
      ]
}
```

Listing 2: Describing services using RAML

```
#%RAML 0.8
title: Person dealing with temperature anomalies
version: v1
baseUri: https://{URL}/{version}/
protocols: [HTTP, HTTPS]
mediaType: application/json


  /api:
    displayName: APIs
    description: Show available APIs
    get:
      responses:
        200:
    /contact:
      /email:
        description: Returns the email address of this
             person
          queryParameters:
            id:
              type: number
          get:
            responses:
              200:
```

V. Two-Phase Discovery Component

To remove the load of handling network connections from discovery component, network discovery module is responsible for registering all devices connecting to a specific domain. Devices can use multiple communication protocols such as Bluetooth, ZigBee, and Wi-Fi to register themselves and obtain a unique identifier. Later, each participating entity including smart devices and humans must submit its description document with the assigned unique identifier. This way, discovery module knows which entities are active and what services are available to be called.

For efficient discovery of things and their available services, a two-phase syntax-based discovery approach will be used. Syntax-based searching methods are fast and do not require any knowledge about the underlying data, because search is performed by matching the data against desired search keywords.

Purpose of the discovery component is not only finding an entity with desired capabilities (e.g capable of measuring temperature and humidity) and desired properties (e.g. located at specific room) but also the required API associated with it. In the first phase, the discovery module will search in TDD repository to find entities matching given search criteria. The way this component is implemented gives the user the option of choosing between exact match or subsume match when submitting a query [16].

Discovered entities may contain group of smart devices having specific properties or can be humans having specific communication details while operating a desired device. After finishing the first step, if the target is finding an API capable of doing specific task, another search is performed on the API Description Documents of devices or humans found in the first phase. We provide a comprehensive discovery example in the next section.

VI. Case Study and Implementation

In this section a case study is described which helps us demonstrate the effectiveness of our proposed framework.

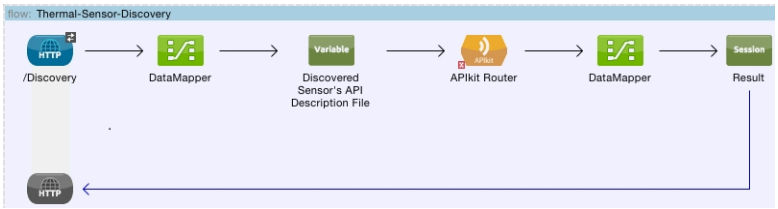

Fig. 2: Thermal sensor discovery flow

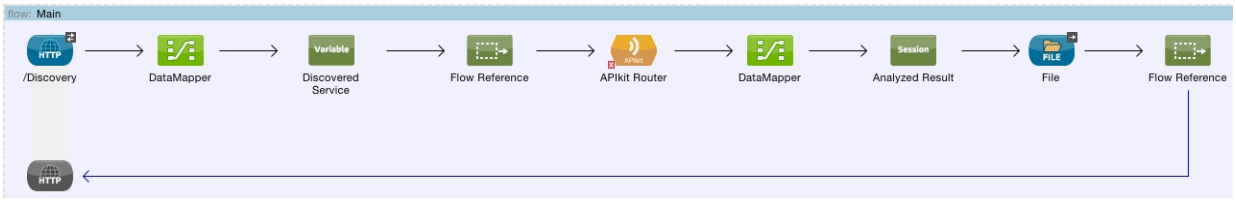

Fig. 3: Anomaly detection and further processing flow

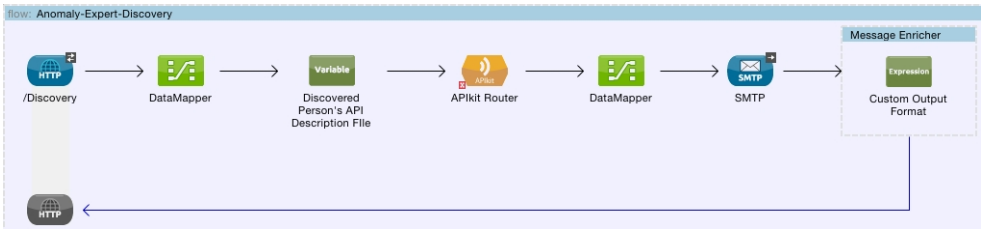

Fig. 4: Anomaly expert discovery flow

The case study is a smart university campus containing large number of sensors including humidity and temperature. The main objective of this case study is to show how a building manager can use our framework to adjust temperature of rooms dynamically based on room temperature, outside temperature and electricity price. Let's consider a case of a server room, so the desired flow has to be able to:

- Locate a device with capability of reading temperature from the server room located in DMD 7.12. The discovered sensor has to be equipped with an API that provides temperature with numeric format and in Celsius degrees.
- Discover and call an API that can detect anomalies of streamed data (in this case, continuous temperature measurement from a thermal sensor) in the output of calling previous API.
- Discover and call an API that can store the anomaly data. These generic APIs that are not device specific, can be implemented using any SOA approach, such as SOAP of REST. A smart interface is responsible for parsing service description files such as WSDL, if required, and finally call the service. Returned result can be mapped to another format or object by defining routines that describe how the mapping between input and output should be performed. This way, chaining service calls and designing flows can be easily achieved.
- Find a device or human who can deal with the abnormal temperature situation.
- Discover and call an API that has the capability of communicating with the discovered human or device.

For demonstrating the realization of how flow orchestration can influence task execution and better integration, we use Anypoint Studio [15], a tool for enterprise service integration to create and execute flows. Each flow consists of several service calls which can contain services from other "things" or be services from outside current domain, such as social media services. Since security is one of the important points of consideration in IoT, service calls can use SSL protocol and security mechanisms such as OAuth for providing confidentiality and proper access control.

For the given case study, end users will be presented with an interface to discover their desired entities by searching through them according to name, type, location, or any other custom defined property. The discovery process is implemented as a RESTful web service and can be extended to look for "things" with APIs containing specific words and input types. When a match is found by searching through API definition files, that service will be called by Flow Execution Engine and the response will be analysed by a smart mapper, responsible for detecting response type. After this step, users can create mappers to extract information from response of a service call, change its format if needed, and then forward it as an input of another service call. This chaining of service calls will make sophisticated operations feasible.

As shown in Figure 2, first the discovery component which is implemented as a RESTful web service is invoked. The parameters used when calling this service depends on what

the user is searching for. In this scenario, the user is looking for a thermal device capable of sensing a specific room's temperature and returning the result in Celsius degrees. Thus, all these information will be sent as input parameters to the discovery engine and the result is analysed and mapped to appropriate format using a data mapper component. Finally, the URL of the sensor's API description document is saved in a local variable.

The next step is calling sensor's service using its API description document, which is in RAML format. This is achieved by using "APIkit Router" tool to parse the RAML file and then invoke the target service. We should mention that the way each RAML file is parsed and its services are invoked in Anypoint Studio, requires additional steps including loading the file and calling any specific service defined in it, which for sake of simplicity are not shown here. Behind the scenes, APIkit Router first loads the configuration settings, giving the location of RAML file for this entity. Then based on the services defined in the RAML file, appropriate stubs are created to connect to the specified endpoint . Finally, the result of calling the web service to sense the room's temperature is mapped to a convenient format and stored in a global session variable.

In Figure 3, same approach for using discovery component is taken to search for an anomaly detection service. The result from calling discovery component is mapped and saved in a local variable and then the first flow is called to get the room's temperature. Later, anomaly detection service is called and the analysed result is stored in a global session variable and a local file. Here we used a pre-defined toolkit to store the file, but same discovery and call procedure can be used to find services providing the option of saving data to any desired file.

Figure 4 depicts the same process used in previous steps to discover a person capable of handling anomalies in data. Again for simplicity, we assume such a person exists, otherwise, we can use condition specifying tools to check if the result from any step is empty or has a specific content. By passing the RAML file containing the API description of how to contact the expert person to APIkit tool and eventually calling the target service, we can then use a SMTP toolkit to notify him by Email.

VII. Conclusions and Future Directions

In this paper, we introduced a framework to describe devices, sensors, humans, and their available services using web API notation and API definition languages. A two-phase discovery approach was also proposed to find entities having certain properties, while having services that match a specific pattern of keywords and input types. Additionally, flow composition was opted and discussed as an appropriate way of integrating "things" and creating sophisticated tasks, thus forming a novel and practical framework for IoT domain.

For future work, we plan to extend current framework to support semantic reasoning and annotations, resulting in more precise query matching rate. Furthermore, considering semantic web and Linked Data with protocols specially designed for IoT like CoAP can be pursued as future directions.

Moreover, since Cloud Computing provides great opportunities for executing flows based on how many resources they need, flow scheduling optimization in Cloud environments, particularly scheduling flows for integrated stream processing is of our interest.

References

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 20132018," June 2014. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] MuleSoft, "Restful application modeling language." [Online]. Available: http://www.raml.org

[4] I. Ishaq, J. Hoebeke, J. Rossey, E. D. Poorter, I. Moerman, and P. Demeester, "Enabling the web of things: facilitating deployment, discovery and resource access to iot objects using embedded web services," *International Journal of Web and Grid Services*, vol. 10, no. 2, pp. 218–243, 2014.

[5] G. K. Teklemariam, J. Hoebeke, I. Moerman, and P. Demeester, "Facilitating the creation of iot applications through conditional observations in coap," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, pp. 1–19, 2013.

[6] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: Scalable cloud services for the internet of things with coap," in *Proceedings of the 4th International Conference on the Internet of Things (IoT 2014)*, 2014.

[7] S. Mayer, D. Guinard, and V. Trifa, "Searching in a web-based infrastructure for smart things," in *Internet of Things (IOT), 2012 3rd International Conference on the*. IEEE, 2012, pp. 119–126.

[8] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services," *Services Computing, IEEE Transactions on*, vol. 3, no. 3, pp. 223–235, 2010.

[9] V. Stirbu, "Towards a restful plug and play experience in the web of things," in *Semantic computing, 2008 IEEE international conference on*. IEEE, 2008, pp. 512–517.

[10] P. Li, J. Dong, J. Wen, and W. Zhou, "A soa-based service discovery framework in internet of things," *Journal of Convergence Information Technology*, vol. 6, no. 9, 2011.

[11] R. Klauck and M. Kirsche, "Bonjour contiki: A case study of a dns-based discovery service for the internet of things," in *Ad-hoc, Mobile, and Wireless Networks*. Springer, 2012, pp. 316–329.

[12] Q. Wei and Z. Jin, "Service discovery for internet of things: a context-awareness perspective," in *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*. ACM, 2012, p. 25.

[13] S. Mayer and G. Basler, "Semantic metadata to support device interaction in smart environments," in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM, 2013, pp. 1505–1514.

[14] Swagger, "A framework for api definition." [Online]. Available: http://swagger.io

[15] MuleSoft, "Anypoint studio." [Online]. Available: http://www.mulesoft.com/platform/mule-studio

[16] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with owls-mx," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 2006, pp. 915–922.