Semantic Web 11 (2020) 949–975
DOI 10.3233/SW-200405
IOS Press

Semantic Node-RED for rapid development of interoperable industrial IoT applications

Aparna Saisree Thuluva [a,b,*], Darko Anicic [a], Sebastian Rudolph [b] and Malintha Adikari [c]
[a] *Corporate Technology, Siemens AG, Munich, Germany*
E-mails: aparna.thuluva@siemens.com, darko.anicic@siemens.com
[b] *Computational Logic Group, TU Dresden, Dresden, Germany*
E-mail: sebastian.rudolph@tu-dresden.de
[c] *Data Engineering & Analytics, TU Munich, Munich, Germany*
E-mail: malintha.adikari@tum.de

Editors: Dhaval Thakker, University of Bradford, UK; Pankesh Patel, Fraunhofer USA – CESE, USA; Muhammad Intizar Ali, Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland; Tejal Shah, Newcastle University, UK
Solicited reviews: Raghava Mutharaju, Knowledgeable Computing and Reasoning Lab, IIIT-Delhi, India; two anonymous reviewers

Abstract. The evolution of IoT has revolutionized industrial automation. Industrial devices at every level such as field devices, control devices, enterprise level devices etc., are connected to the Internet, where they can be accessed easily. It has significantly changed the way applications are developed on the industrial automation systems. It led to the paradigm shift where novel IoT application development tools such as Node-RED can be used to develop complex industrial applications as IoT orchestrations. However, in the current state, these applications are bound strictly to devices from specific vendors and ecosystems. They cannot be re-used with devices from other vendors and platforms, since the applications are not semantically interoperable. For this purpose, it is desirable to use platform-independent, vendor-neutral application templates for common automation tasks. However, in the current state in Node-RED such reusable and interoperable application templates cannot be developed. The interoperability problem at the data level can be addressed in IoT, using Semantic Web (SW) technologies. However, for an industrial engineer or an IoT application developer, SW technologies are not very easy to use. In order to enable efficient use of SW technologies to create interoperable IoT applications, novel IoT tools are required. For this purpose, in this paper we propose a novel semantic extension to the widely used Node-RED tool by introducing semantic definitions such as iot.schema.org semantic models into Node-RED. The tool guides a non-expert in semantic technologies such as a device vendor, a machine builder to configure the semantics of a device consistently. Moreover, it also enables an engineer, IoT application developer to design and develop semantically interoperable IoT applications with minimal effort. Our approach accelerates the application development process by introducing novel semantic application templates called Recipes in Node-RED. Using Recipes, complex application development tasks such as skill matching between Recipes and existing things can be automated. We will present the approach to perform automated skill matching on the Cloud or on the Edge of an automation system. We performed quantitative and qualitative evaluation of our approach to test the feasibility and scalability of the approach in real world scenarios. The results of the evaluation are presented and discussed in the paper.

Keywords: Industrial Internet of Things, Web of Things, Node-RED, iot.schema.org, semantic interoperability, edge computing, skill matching

1. Introduction

The vision of Internet of Things (IoT) is to digitize the physical world and offer new classes of applications that are based on this digitization. In this process,

[*] Corresponding author. E-mail: aparna.thuluva@siemens.com.

1570-0844/20/$35.00 © 2020 – IOS Press and the authors. All rights reserved

various sensors and actuators are attached to physical things in order to sense and interact with the physical environment. IoT is used to interconnect devices and bring added-value in many domains such as personal things: e.g., health monitors, wearable devices etc., in automation systems such as Building Automation System (BAS), Industrial Automation System (IAS), plants and manufacturing facilities. IoT used in complex domains such as automation systems BAS, IAS, plants and manufacturing facilities is called Industrial Internet of Things (IIoT) [15]. IIoT envisions to create seamless integration of things, where things "know" what they should do, and interact accordingly by exchanging information. In order to fulfill this vision, IAS should be flexible and the process of developing new applications on them should be simplified. That is, we should be able to develop new applications on them faster, with low-cost, less effort. While new applications are required to enable interaction between devices to automate processes, for predictive maintenance, early fault detection, mass customized production, etc., that keep a production process alive and controls the surrounding environment. Nevertheless, a key challenge should be addressed in order to enable rapid IIoT application development: vast amounts of data is produced by sensors and actuators on an IAS and BAS, which is unstructured. That is, the Knowledge about the data (metadata) is not available. The Knowledge about the data should be described consistently using common and standardized domain semantic models, in order to process the data efficiently and bring added value. Moreover, it should also accelerate and simplify the application development on an IAS or BAS, since the applications can be developed against the common and standardized semantics. Application development here means to design, develop, configure and deploy new applications on an Automation System (AS).

The methods employed today for the development of a new application on a complex IAS or BAS requires a lot of manual effort, they are time-consuming and expensive, since the state of the art IAS and BAS are engineered with certain applications in mind. Once these systems are engineered and deployed, it is not feasible to develop and deploy new types of applications on them [2]. This would require very high effort and time by expert engineers who have knowledge about the capabilities and configurations of the complex machinery on the system. The data about the complex machinery is described using heterogeneous semantics. Therefore, even an expert might struggle to understand the non-uniform data coming from different sources. He then uses a special engineering IDE (e.g., 4DiAC [31], TIA portal[1]) to develop a Function Block, test it and deploy it with considerable effort on control devices that are controlling the machinery such as a Programmable Logic Controller (PLC). Considering all the above steps, the development of new applications on state of the art automation systems is a high-effort and time consuming process. With the introduction of Service Oriented Architecture (SOA) in industrial automation, the application development on IAS can be simplified significantly, since every device can be exposed as a Web service, an application can be developed as Web service choreography [14]. Furthermore, with the advent of IoT, industrial devices are being equipped with micro-controllers that enable the devices to connect to the internet, which led to IIoT. In IIoT, application development can be further simplified: now applications can be developed as IoT orchestrations using IoT application development tools such as Node-RED. Nevertheless, still an expert engineer with domain knowledge is required to develop applications, since the knowledge about the capabilities of the devices, their configurations and, commissioning information is described using heterogeneous semantics. Therefore, the devices are not interoperable, only experts can understand their capabilities. Applications are being developed by experts, and they are bound to the devices from a specific vendor and a specific platform. Thus, the applications are not interoperable between diverse vendors, platforms and device ecosystems. For every platform and vendor an expert needs to create applications, which is time consuming, expensive and requires a lot of effort. Several domain semantic models such as OPC UA companion specifications are available to address this problem. However, they are hard to use.

1.1. Vision

The application development process can be revolutionized by using modern Web technologies such as Semantic Web (SW) and Web of Things (WoT) technologies. In this paper, our vision is to develop a novel approach for rapid development of interoperable IoT applications using these technologies, where rapid means faster application development with low-cost and less manual effort. As we discussed before, only expert engineers can develop applications on complex

[1] www.siemens.com/global/en/home/products/automation/industry-software/automation-software/tia-portal.html

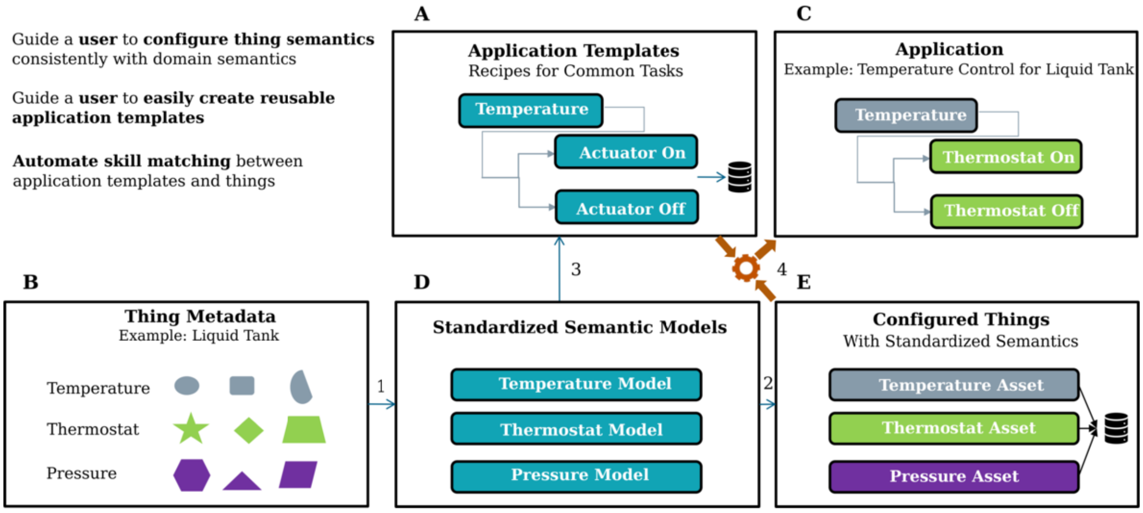


Fig. 1. Overview of rapid IIoT application development approach.

automation systems and the availability of the experts is limited. In addition to experts, if we enable various IoT stakeholders such as Web developers, IoT application developers and machine vendors etc. in the application development process then we can realize the vision of rapid application development. The vision is depicted in Fig. 1, which is to design and develop composable and customizable applications on existing things quickly and easily. Figure 1, Box C shows an application to control the temperature of liquid in a tank by measuring the current temperature of the liquid and controlling it using a thermostat. Such applications are required for common automation tasks. In order to speed up the application development process, it is desirable to create standardized application templates as shown in Fig. 1, Box A. These templates can be instantiated with things on an IAS or a BAS as shown in Fig. 1, Box B. However, instantiating an application template on things with matching skills is not easy, since, the complex machinery on an AS is usually described with heterogeneous semantics and considerable manual effort is required to discover things with matching skills for an application template. Therefore, to simplify and accelerate application development, skill matching needs to be simplified. Our goal is to automate the skill matching process. For this purpose, we propose to use common and standardized semantic models as shown in Fig. 1, Box D. Using these semantic models our approach can be described as follows:

1, 2. Configure the attributes, capabilities, configurations and commissioning information of the things [10,16,32,39] consistently using common and standardized semantic models as shown in Fig. 1, Box E.
3. Describe the application templates also using common and standardized semantic models as shown in Fig. 1, Box A.
4. Automate the skill matching between configured things and application templates to generate customized applications with matching things on an AS, as shown in Fig. 1, Box C.

In our approach, we choose the semantic models described using SW knowledge representation formalisms such as RDFS, OWL, etc. Using SW formalisms, the knowledge becomes machine interpretable and SW technologies offer efficient query engines and reasoners that can be employed for automated skill matching between the application requirements and existing things.

The application templates in our approach are called *Recipes* [34]. A *Recipe* describes the skills or capabilities of things required to run an application and the data-flow between the things in a machine-interpretable format. It can be instantiated any number of times with the existing things from any platform, ecosystem or vendor. Since Recipes and things are described using platform-independent and format-neutral semantics, this enables semantic interoperability. It enhances the re-usability of applications, which in turn lowers the cost and time required for application development. In addition to this, the complex application development tasks are automated using Recipes, which enables non-expert engineers to design, develop, configure and deploy IIoT applications.

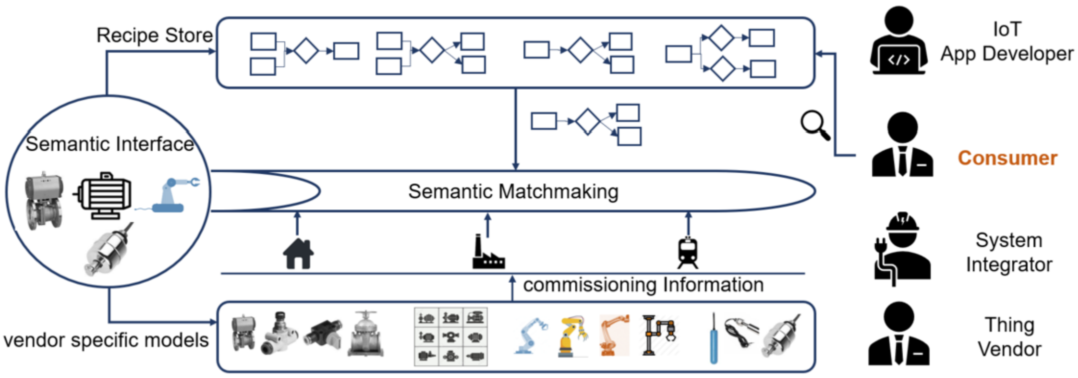


Fig. 2. Overview of rapid IIoT application development approach.

Figure 2 represents the semantically interoperable IIoT ecosystem that we propose in this paper. Here, the domain experts and semantic experts create the semantic interfaces for IIoT things. These semantic interfaces are then used by various IoT stakholders to achieve rapid application development. A machine vendor can use these interfaces to describe the semantics of their things and offer them to their customers. A system integrator can then add the commissioning information to the semantic description provided by the vendor, where a thing is deployed in a factory or building or a vehicle etc. Such a semantic description of a thing can then be stored in a knowledge store. On the other hand, a Web developer or an IoT application developer can use the semantic interfaces and design an IoT application template, that is, a Recipe. These Recipes can be stored in a Recipe store which is a knowledge store where they can be discovered and reused. In such a scenario, a customer can simply login to the Recipe store, browse for the required Recipes and then he can easily instantiate the application and deploy it on his workstation, since the complex application development task such as the skill matching is automated in our approach. Moreover, Recipes reduce the effort of application development, since, the applications need not be developed from scratch every time.

1.2. Building blocks

In this paper, we chose the following building blocks to realize our rapid IIoT application development approach:

1. IoT semantic models such as **iot.schema.org**, which describe affordances and data schema of things in several IoT domains. iot.schema.org semantic models act as a common and standardized semantic models in our approach;
2. novel engineering and IoT application development tools such as **Node-RED** that simplify the IoT application development process through visual programming. Node-RED acts as a tool that enables a non-expert to do complex application development tasks easily;
3. the upcoming **W3C Web of Things (WoT) standard** which provides **Thing Descriptions (TD)** to describe things in a format neutral and platform-independent manner.

There is no semantic interoperability in Node-RED, in the current state, which limits the usage of applications (developed in Node-RED) across diverse IoT platforms. In this work, our goal is to enable semantic interoperability in Node-RED by extending it with iot.schema.org semantics. As mentioned earlier, it empowers machine builders, device vendors to mark-up their things with iot.schema.org semantics and generate semantically enriched W3C WoT TD for their things. Further, it allows applications to be easily implemented for things across diverse device ecosystems and platforms. Ultimately, it opens up a huge market for interoperable and highly re-usable IoT applications.

1.3. Contributions

Our main contribution in this work is a novel approach to build a semantically interoperable ecosystem for IIoT that can be easily used by different IoT stakeholders for rapid application development. For this purpose, we propose the usage of lightweight IoT semantic models and a novel way of using them as graphical templates in an IoT application development tool. In this way, all the IoT stakeholders can easily use the semantics, since, various stakeholders in IoT may not be experts in SW technologies. We further developed an approach to integrate semantic querying into the tool for the purpose of automated skill matching during IoT application development. These approaches are deployed in real industrial environment on an IAS, where we conducted qualitative and quantitative experiments. We will present the results of the experiments in this paper.

1.4. System architecture

Figure 3 represents the system architecture of our approach. In our approach there are three levels. They are the *Field level*, the *Edge or Factory level* and the *Cloud level* as shown in Fig. 3. The data flow occurs in the following manner: (1) heterogeneous data from complex ASs from field level is sent to the edge/factory level; (2) the structuring of the data can be done on the Edge or, alternatively the Edge collects the data and sends it to the Cloud; (3) the unstructured data is sent to the Cloud where semantically extended Node-RED is running. There the data is structured uniformly (this approach presented in Section 4 & Section 5); (4) the structured data can then be stored in a knowledge graph in a machine interpretable format. It enables efficient discovery of data (presented in Section 6); (5) The standardized semantic models are also used to design Recipes as Node-RED flows (the approach presented in Section 7); (6) the Recipes thus created can also be stored in a knowledge graph and they can be discovered, reused, extended and shared (presented in Section 7); (7) we enable automated skill matching using Recipes, which can be done either on the Cloud or on the Edge (this approach is presented in Section 7). Finally the applications can be deployed on the Cloud or optionally on the Edge.

1.5. Paper structure

This paper is structured as follows: In Section 2, we provide an overview of the background and related work. In Section 3, we will briefly present an industrial use case, which will be used throughout the paper to explain all the concepts developed in this work.

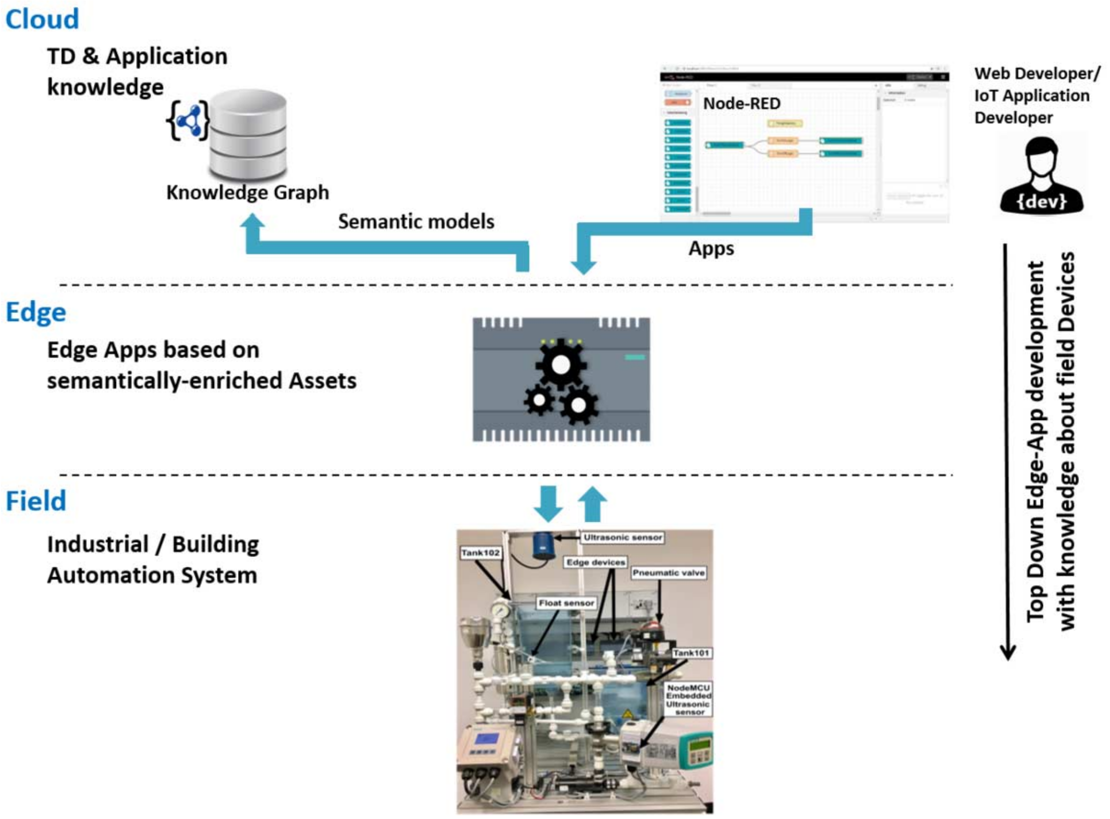

Cloud
TD & Application knowledge
Knowledge Graph
Semantic models
Node-RED
Apps
Web Developer/ IoT Application Developer
{dev}
Edge
Edge Apps based on semantically-enriched Assets
Field
Industrial / Building Automation System
Tank102
Ultrasonic sensor
Edge devices
Pneumatic valve
Float sensor
Tank101
NodeMCU Embedded Ultrasonic sensor
Top Down Edge-App development with knowledge about field Devices

Fig. 3. System architecture of interoperable IIoT application development approach using semantic Node-RED.

We describe our approach in Section 4, Section 5, Section 6 and Section 7. In Section 8, we will discuss the evaluation performed to analyze the feasibility and scalability of our automated skill matching approach in real world use cases, and our qualitative evaluation of the semantically extended Node-RED tool. In the same section, we also evaluate the semantic Node-RED tool against the existing IoT application development tools, industrial engineering tools etc. Section 9 presents the details about implementing the Asset Administration Shell (AAS) by employing W3C WoT technologies and semantic Node-RED tool. Section 10 briefly describes the industry challenges faced and lessons learned. In Section 11, we will conclude the work, present the limitations of the approach and discuss future research directions.

2. Background & related work

2.1. W3C web of things working group

is developing a standard to create interoperability between physical things on the Web. For this, WoT Group is developing a protocol binding to enable interoperability between various protocols such as OCF, OPC-UA [23], BacNet [25] and so on. Apart from the protocol binding, WoT Group also proposes Thing Description, which is a platform independent description of a physical device. Thing Description describes a device in terms of its interactions such as *Properties*, *Events* and *Actions*. A Thing Description is serialized in JSON-LD [9] format. Further on, Thing Description uses JSON Schema [40] to model syntactical constraints on data. JSON Schema provides simple data types such as *integer*, *number*, *string*, *boolean* and complex data types such as *object*, *array* and *enumeration*.

2.2. IoT ontologies

In order to provide semantic discovery and interoperability between devices, Thing Description is supposed to be extended with external ontologies and schemas [5,30]. Many ontologies are developed for IoT. Domain-independent ontologies such as W3C Semantic Sensor Networks (SSN) [8], M3 [11] to model sensors, their observations and their domain. QUDT [13] to model physical quantities and Units of measurement. WGS84 [4] describes concepts to define spatial features of IoT things. IoT Ontology [21] which models IoT entity features required for their automated deployment. There also exists several domain ontologies such as SAREF[2] for Smart appliances, Brick [1] for Building Management Systems (BMS) domain. There exists eCl@ssOWL ontology [12] for industry domain and so on. A TD can be semantically enriched by marking up with the terms defined in these ontologies. More recently, community work on iot.schema.org (currently referred to as iotschema.org[3]) has started which provides lightweight RDFS semantics to expose capabilities of a device in order to simplify IoT application development. iot.schema.org Capability model is aligned with WoT TD model [30]. In order to model domain features of a physical device, iot.schema.org normalizes semantics from existing standards such as OneM2M,[4] OpenT2T,[5] OCF,[6] IPSO Objects.[7] Moreover, the semantics of Capability model can be further extended with existing IoT semantics. For example, Capability model can be extended with Feature of Interest model defined in W3C SSN/SOSA ontology.

2.3. iot.schema.org

More recently, community work on iot.schema.org (currently referred to as iotschema.org[8]) has started, which provides lightweight RDFS semantics to expose capabilities of a thing in order to simplify IoT application development. **iot.schema.org** is a publicly available repository of domain-specific semantic definitions for connected things.[9] It is being developed as an extension to the widely used schema.org to enable descriptions of things and their data in the physical world. iot.schema.org definitions provide a protocol independent and format neutral way for applications to understand the affordances of, and data provided by physical things. These definitions are used to annotate instances of physical things and their data using simple markup with common formats like HTML, JSON and web linking formats. Device vendors can use iot.schema.org to publish protocol-neutral defini-

[2] https://w3id.org/saref
[3] http://iotschema.org/
[4] http://www.onem2m.org/
[5] https://github.com/openT2T/translators
[6] https://oneiota.org/
[7] https://github.com/IPSO-Alliance/pub/tree/master/reg
[8] http://iotschema.org/
[9] https://github.com/iot-schema-collab/intro-materials/blob/master/iotschema-intro-overview.pdf

```
{   "@context" : [{
        "schema": "http://schema.org/",
        "iot": "http://iotschema.org/"  }],
    "@id": "iot:AirConditioner",
    "rdfs:subClassOf": { "@id": "iot:Capability" },
    "rdfs:label": "Air conditioning Capability",
    "iot:domain": [{"@id": "iot:Industry"}, {"@id": "iot:Building"}],
    "iot:providesInteractionPattern": [{
        "@id": "iot:Temperature",
        "@id": "iot:TargetTemperature",
        "@id": "iot:SwitchStatus",
        "@id": "iot:TurnOn",
        "@id": "iot:TurnOff",
        "@id": "iot:RunMode",
        "@id": "iot:CountDown",
        "@id": "iot:WindStrength"}] }
```

Listing 1. Specification of iot.schema.org AirConditioner capability

tions for their things to enable Web scale adoption. Application providers can use iot.schema.org to make their applications portable across platforms.

The iot.schema.org Capability model is aligned with the WoT TD model [30]. In order to model domain semantics of a physical device, iot.schema.org normalizes semantics from existing standards such as OneM2M,[10] OpenT2T,[11] OCF,[12] IPSO Objects.[13] Moreover, the semantics of the Capability model can be further extended with existing IoT semantics. For example, the iot.schema.org Capability model is extended with the Feature of Interest model defined in W3C SSN/SOSA ontology [8].

An iot.schema.org Capability provides a semantic description of a physical thing in terms of its interaction patterns (Properties, Events and Actions) and their input and output data schemas. Let us consider an example iot.schema.org Capability defined for an air conditioner. The Capability is created referring to OCF, OneM2M and IPSO description of an air conditioner. Due to space constraints, we only present a part of the Capability in Listing 1. The complete specification of the AirConditioner Capability, its interaction patterns and data can be found in the GitHub repository mentioned above. Among others, the AirConditioner Capability provides the following interaction patterns (1) *TurnOn*, an action to turn on an air conditioner; (2) *TurnOff*, an action to turn off an air conditioner; (3) *SwitchStatus*, a property to check the current status of an air conditioner; (4) *RunMode*, an action to set the operating mode of an air conditioner to *FanMode*, *CoolMode*, *DryMode* or *EnergyOrPowerSaverMode*. These interaction patterns are further described in terms of their input and/or output data schema as shown in Listing 2.

2.3.1. *Data description*

The Data schema of an interaction pattern is well-defined in terms of the value type, allowed values for the data, units of measurement, minimum and maximum range of the data etc. iot.schema.org uses the W3C standard Shapes Constraint Language (SHACL) [18] to describe an interaction pattern and its data schema. This is because the interaction patterns and their data schema can be expressed as RDF shape constraints, which enables a machine to validate input or output data associated with an interaction. Listing 2 shows the definition of data for the *Temperature* interaction pattern of the AirConditioner Capability. The *Temperature* property has output data called *TemperatureData*. The *TemperatureData* shape describes that the *Temperature* interaction pattern provides a value of type *float*. Minimum and maximum range for *Temperature* should be specified in *float*. Moreover, the unit of measurement for *Temperature* should be either *Celsius*, *Fahrenheit* or *Kelvin*.[14] Data description for all interaction patterns of AirConditioner Capa-

[10] http://www.onem2m.org/

[11] https://github.com/openT2T/translators

[12] https://oneiota.org/

[13] https://github.com/IPSO-Alliance/pub/tree/master/reg

[14] shorturl.at/ckM14

```
 iotsh:TemperatureDataShape a sh:NodeShape ;
 sh:targetClass iot:TemperatureData ;
 sh:and (
[ sh:property [
 sh:path schema:propertyType ;
 sh:minCount 1 ;
 sh:maxCount 1 ;
 sh:datatype xsd:float ; ]; ]
[  sh:property [
 sh:path schema:minValue ;
 sh:minCount 1 ;
 sh:maxCount 1 ;
 sh:datatype xsd:float ; ]; ]
[  sh:property [
 sh:path schema:maxValue ;
 sh:minCount 1 ;
 sh:maxCount 1 ;
 sh:datatype xsd:float ; ]; ]
[ sh:property [
 sh:path schema:unitCode ;
 sh:minCount 1 ;
 sh:maxCount 1 ;
 sh:in ( iot:Celsius iot:Fahrenheit iot:Kelvin ) ; ]; ] ).

 iotsh:TemperatureShape a sh:NodeShape ;
 sh:targetClass iot:Temperature ;
 sh:and (
[ sh:property [
 sh:path iot:providesOutputData ;
 sh:minCount 1 ;
 sh:maxCount 1 ;
 sh:node iot:TemperatureData ; ]; ] ).
```

Listing 2. SHACL shape definition for iot.schema.org temperature interaction pattern

bility can be found in iot.schema.org GitHub repository.

In this manner, iot.schema.org uses the RDFS semantics to model capabilities and RDF SHACL shape constraints to model interaction patterns and data schema. Such Capability semantic descriptions can be used to model the affordances and data schema of physical things in the real world. However, an iot.schema.org Capability defines a class of physical things, not all the physical things belonging to that class can fulfill a Capability, interaction pattern and data schema definition. That is, every device vendor should configure/customize a capability definition, in order to fit to his physical thing specification. Later in this paper, we will present how shape constraints on an interaction pattern and data schema can be used to configure an iot.schema.org definition according to a physical thing specification.

However, semantic descriptions are difficult to use by Web developers, IoT application developers, device vendors and machine builders. The existing approaches to enable usage of semantic models such as the schema.org approach provides a Web page which describes a class in terms of its relations and their expected values. Moreover, the schema.org also provide examples on how to use a class. This is proven to be a good approach to use semantic models by Web developers. However, this approach is error-prone, since a Web developer should mark-up a Web page using the semantic model manually and the process is not validated. Therefore, we should simplify this process further to enable non-experts to use semantic definitions to mark-up their things or WoT applications.

2.4. Web service composition

Service composition is a process of discovering the required services, reserving them and connecting them to each other. There has been significant work done in this area where the initial methods such as Web Service Description Language (WSDL) [6] focused only on the syntactical information of services for composition. Later came the approaches that took semantic information about services into consideration during composition. They are approaches using OWL-S [24] ontology which is a W3C recommendation that can be used together with OWL to define semantics of data and operations of Web services. Other methods such as WSMO standard [22], SAWSDL [20] came into existence. However, in more recent years, the design of Web services and APIs followed more often the REST principles, instead of WSDL and SOAP based Web services. Then came the approaches such as hRESTs [19], RESTdesc [37] into existence. With the advent of IoT some low-code application development tools such as Node-RED [28], IFTTT [29], etc., came into existence for the composition of IoT things and services. A detailed related work about this topic can be found in this paper [34]. In this work we developed a novel approach of Web service composition tailored for Web of Things. The IoT ecosystems such as Amazon IoT Things Graph, Alibaba cloud IoT platform, Microsoft Azure IoT describe their things in terms of Properties (or services), Events and Actions (or commands). These interactions are being normalized by W3C Web of Things Thing Description. Our Recipe mechanism is a novel approach for composition of Web of Things interactions. Some of the main differences between existing Web service composition methods and Recipes is that, application logic is not part of a Recipe description, which makes it simple and lightweight. Moreover, a Recipe is a reusable template for the composition of things. One of the main aims of Recipe is to make it easily usable by IoT stakeholders such as machine vendors, system integration, IoT application developers, Web developers, etc who may not be experts in using semantic technologies. Therefore, it is embedded into existing IoT application development tools such as Node-RED, instead of developing yet another new tool for semantic-based IoT service composition.

3. Use case

We consider the industrial integration use case defined by the W3C WoT community to demonstrate the rapid WoT application development approach developed in this work.[15] The use case is taken from one of the W3C WoT PlugFests where participants from different companies participated and things came from diverse vendors such as Panasonic, Fujitsu, Festo, KETI, Siemens etc. The use case demonstrates the complexity of real world scenarios as the things used in this use case belong to diverse device ecosystems, different vendors and are heterogeneous. This demonstrates how W3C WoT technologies can be used to bind things from diverse ecosystems to uniformly interact with the things to create value-added services using the things. In addition, we will also demonstrate how domain semantic interoperability will enable rapid development of added-value services on such systems.

The use case is to “automatically alert and protect citizens when a chemical plant has an accident”. The scenario is the following: The KETI environment sensor is capable of measuring air quality by measuring the oxygen level. When a low oxygen condition is detected, then, the connected devices take the appropriate action to protect and alert citizens, such as draining a tank in the factory, turning off all air-conditioners, flash alert and warning lights, publish alert messages and make voice announcements.

The use case is demonstrated using things provided by different vendors such as the following: an environment sensor from KETI, a FESTO PA workstation with tank liquid level control system deployed at Siemens which is shown in Fig. 8, warning lights provided by Fujitsu, air conditioners from Panasonic. All the things are connected to the Web over the Oracle cloud. In the next sections, we will see how semantically interoperable WoT application can be developed for this use case using our approach.

4. Semantic configuration nodes in Node-RED

One of the motivation behind extending Node-RED with semantics is to enable a user who is not an expert in SW technologies such as a thing vendor, an engineer, etc., to describe the semantics of a thing, its

[15] https://github.com/w3c/wot/blob/master/plugfest/2018-lyon/Scenarios.md

affordances and data schema. A machine-interpretable description of a thing plays a key role in automating the engineering process of an IAS, in automating the application development on an IAS and for plug-and-play functionality of things.

In our previous work [33], we described how an iot.schema.org semantic model can be configured or customized using Shape constraints to describe thing variants in a class of physical things. In this work, we will present a novel approach to model thing variants in Node-RED. Node-RED has always been used to create run-time IoT applications. For the first time, we introduce a novel feature to use Node-RED for design purposes, that is, for tasks such as semantic configuration of skills or capabilities of a thing, designing semantically interoperable WoT application templates, etc. In order to do this, we introduced standardized semantic definitions provided by iot.schema.org community into Node-RED in the following manner. Every interaction pattern and its data schema defined in iot.schema.org is a Node-RED node in our approach. We call these nodes *iotschema nodes*. This is the first step in our approach as shown in step 1 of Fig. 1. An interaction pattern with its data schema is an atomic capability of a thing. A complex thing (physical or virtual) can be described using one or more iotschema nodes in Node-RED. These semantic definitions of things can then be used for multiple purposes, which we will explain throughout the paper.

4.1. *Usage of iotschema nodes*

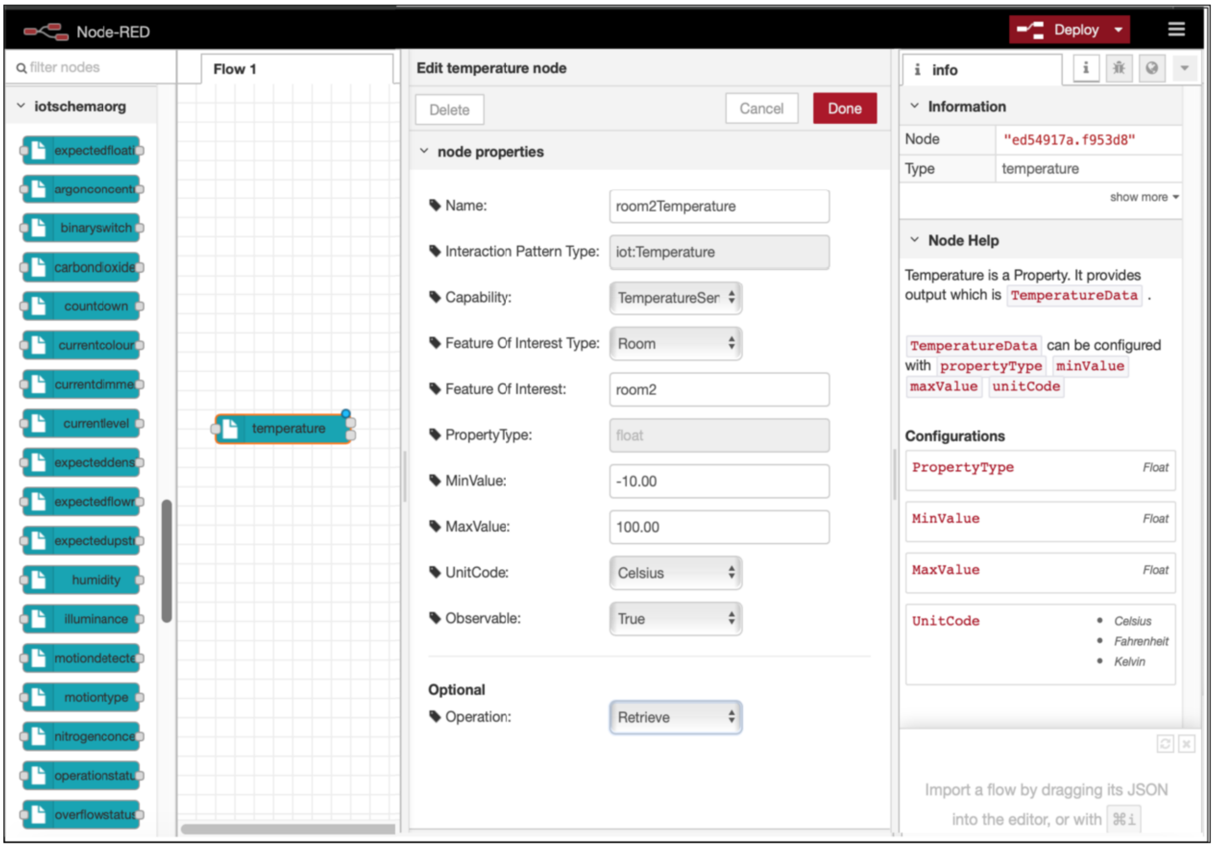
An iotschema node in Node-RED represents a SHACL shape defined for an interaction pattern in iot.schema.org. Listing 2 shows the SHACL shape for a Temperature interaction pattern. The shape specifies that the value type of a temperature data should be float, the minimum and maximum scale of the temperature data should be specified and it should also be a float value. Further on, the unit of measurement of temperature data should be defined and it should be either *iot:Celsius*, *iot:Fahrenheit* or *iot:Kelvin* as described by iot.schema.org. Such an interaction pattern definition can be exposed as a node in Node-RED. These nodes correspond to Box D, that is standardized semantic models in Fig. 1. A user can configure the parameters of these nodes and its data by giving input. The process of giving input is guided by the semantic definition of an interaction pattern and it is validated by its SHACL shape. Figure 4 shows an iotschema node and its configuration attributes in Node-RED. The configuration attributes of an iotschema node are the parameters that are configurable by a user to describe his thing specifications. There are several attributes that a user can configure for a thing's interaction pattern and its data. They are the following:

1. *Capability*: the capability or skill of a thing;
2. *Feature Of Interest Type*: the type of entity whose quantity is being observed or actuated [8]. e.g., a room, a tank, a door etc;
3. *Feature Of Interest*: an entity instance whose quantity is being observed or actuated [8]. e.g., room 1, tank 2 etc;
4. *MinValue*: if applicable, minimum scale of a quantity being observed or actuated by a thing's interaction;
5. *MaxValue*: if applicable, maximum scale of a quantity being observed or actuated by a thing's interaction;
6. *UnitCode*: if applicable, unit of measurement of a quantity being observed or actuated by a thing's interaction;

The user's input for configuration attributes of an iotschema node are used to update the SHACL shape of an interaction pattern according to a thing's specification. Listing 3 shows a snippet of the shape updated according to a user's specification for a temperature sensor. The shape states that the temperature sensor provides temperature data which is a **float** and it can measure temperatures between **0.00** and **100.00 degrees Celsius**. Such a shape can be used to generate a semantically marked-up description of a thing for a specific ecosystem such as W3C WoT, OPC-UA [23] etc. In the future, we assume that a device vendor can ship a thing with its semantic (machine-interpretable) description and an engineer or a system integrator can easily add commissioning information to the semantic description of a thing. In order to simplify the job of a device vendor, an engineer, a machine builder, our tool supports them to provide a vendor-independent and format-neutral semantic description for things easily using semantic configuration nodes provided in Node-RED.

4.2. *Generation & installation of iotschema nodes*

The generation of iotschema nodes in Node-RED is a simple process. Once a semantic expert has modeled a SHACL shape for an interaction pattern and its data schema in iot.schema.org, he or she can generate a corresponding iotschema node by executing a Node.js



Edit temperature node

node properties

Name: room2Temperature

Interaction Pattern Type: iot:Temperature

Capability: TemperatureSer

Feature Of Interest Type: Room

Feature Of Interest: room2

PropertyType: float

MinValue: -10.00

MaxValue: 100.00

UnitCode: Celsius

Observable: True

Optional

Operation: Retrieve

Information

Node	"ed54917a.f953d8"
Type	temperature

show more

Node Help

Temperature is a Property. It provides output which is TemperatureData .

TemperatureData can be configured with propertyType minValue maxValue unitCode

Configurations

PropertyType	Float
MinValue	Float
MaxValue	Float
UnitCode	Celsius Fahrenheit Kelvin

Import a flow by dragging its JSON into the editor, or with ⌘i

Fig. 4. Configuration parameters of iot.schema.org *temperature* interaction pattern node.

```
iotsh:TemperatureDataShape a sh:NodeShape ;
 sh:targetClass iot:TemperatureData ;
 sh:and (
[ sh:property [
 sh:path schema:propertyType ;
sh:datatype xsd:float ;
 sh:minInclusive 0.0 ;
 sh:maxInclusive 100.0 ; ]; ]
[ sh:property [
 sh:path schema:unitCode ;
sh:hasValue iot:Celsius ; ]; ] ).
 iotsh:TemperatureShape a sh:NodeShape ;
 sh:targetClass iot:Temperature ;
 sh:and (
[ sh:property [
 sh:path iot:providesOutputData ;
 sh:minCount 1 ;
 sh:maxCount 1 ;
 sh:node iot:TemperatureData ; ]; ] ).
```

Listing 3. Temperature shape customized according to a vendor specification

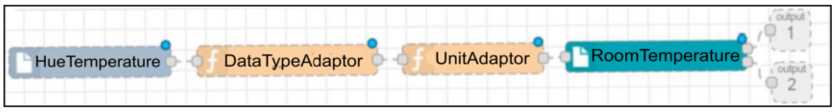


Fig. 5. Semantic integration of a temperature thing node with iotschema temperature node.

script. The script and iotschema nodes for existing iot.schema.org interaction pattern semantic definitions are open-source.[16] The script generates iotschema nodes and installs them in Node-RED. A user can download the generated nodes from the open source *iotschema-node-red* project and install them in their Node-RED. The detailed instructions on how to install the nodes can be found in the project GitHub repository.

5. Semantic integration of existing IoT things

There exist several IoT things from diverse vendors and ecosystems such as sensors or actuators (for example: Philips HueMagic,[17] Amazon Alexa,[18] Google Home kit,[19] Xiaomi smart home,[20] etc.) available in Node-RED. W3C WoT things can also be easily represented as nodes in Node-RED. Furthermore, there exists several existing things on the state of the art automation systems. All these things have vendor and/or platform specific data formats and heterogeneous semantics. In the current state, they are not interoperable with each other. In order to create interaction between them, a gateway must be developed. However, if we can integrate these diverse things using common and standardized semantic models, then we can address the semantic interoperability problem in Web of Things. Moreover, this would accelerate WoT application development.

Our semantic integration approach using iotschema nodes in Node-RED addresses this problem. The purpose of iotschema nodes is not only to create semantic descriptions of new IoT things, but the nodes can also be used for semantic integration of existing IoT things on the stare of the art automation systems. In this section, we explain how semantic interoperability among diverse IoT platforms and vendors can be achieved with semantically extended Node-RED.

In Node-RED, an IoT thing is represented by a node. A thing can be observed or actuated using its thing node. Semantic integration of diverse things is done using a thing node and a corresponding iotschema node to describe its semantics. It is done by wiring an existing thing node with a corresponding iotschema node and configuring the iotschema node according to the specification of the existing thing as shown in Fig. 5. However, several aspects should be taken into consideration for the integration of existing IoT things with the iot.schema.org semantics. Especially, the input and output data schema of a thing should be adapted to be compliant with the iot.schema.org specification. That is, value type, encoding format, and unit of measurement of data of an existing thing should be adapted as prescribed by a corresponding iotschema semantic model. For example, if a temperature thing gives *integer* value as output, however, iotschema temperature interaction pattern prescribes that the temperature data should be *float*, the output of the temperature thing node should be adapted from integer to float, in order to integrate the thing's semantics with iot.schema.org's temperature definition.

Therefore, it may require a few adaptations to integrate an existing IoT thing semantics with a corresponding iot.schema.org semantic definition. For this purpose, we offer an adaptation API to adapt data formats from the diverse IoT ecosystem or diverse serialization formats to the iot.schema.org data format. The adaptation API is also offered as a set of nodes in Node-RED. For example: we provide nodes to convert a data from integer to float or from float to double. We provide nodes to convert data in string format to JSON or vice verse. We provide nodes to convert data from one unit of measurement to another. A user should use one or more adaptation nodes to integrate an existing IoT thing's data with the iot.schema.org specification as shown in Fig. 5. For this purpose, a user should carry out the following steps:

1. He or she creates a flow by wiring a thing node with one or more adaptation nodes and then by

[16] https://github.com/iot-schema-collab/iotschema-node-red
[17] https://flows.nodered.org/node/node-red-contrib-huemagic
[18] https://flows.nodered.org/node/node-red-contrib-alexa-home
[19] https://flows.nodered.org/node/node-red-contrib-google-home-notify-volume-adjustable
[20] https://flows.nodered.org/node/node-red-contrib-xiaomi-smart-home

connecting the adaptation nodes with iotschema node.

2. He then configures the iotschema node according to the specification of the thing.
3. The iotschema node gives two outputs. The first output is the SHACL shape, which is configured according to the thing's specification. The second output is the run-time value given as output by the thing (if that IoT thing gives output).
4. For convenience, a user can save the flow he created using a thing node, adaptation nodes and iotschema node as sub-flow in Node-RED, and later he can use it as a single node during application development.
5. The semantic description of an IoT thing thus created can be stored in a knowledge graph for discovery.

In this way, using our approach semantic integration of heterogeneous IoT things can be easily achieved even by non-experts in semantic technologies.

6. Generation of semantic thing descriptions

As mentioned in the previous sections, the configured SHACL shapes generated in Sections 4 & 5 can be used to generate W3C WoT Thing Descriptions (TDs). According to the state of the art, TDs are modeled manually and semantic enrichment of TDs is also done manually which is a time-consuming and error-prone process, since the common practice is to manually write a TD, find the required semantic descriptions from iot.schema.org website and edit the TD manually to annotate it. In order to overcome these problems, we developed an approach to automate the process of creation and semantic enrichment of TDs. In our previous work [33], we described an algorithm to generate a semantically-enriched Thing Description (TD) from configured shapes of iotschema interaction patterns. However, the approach is still hard to use for users who are not experts in SW technologies. Therefore, we integrated the algorithm into Node-RED to enable a non-expert to easily create a semantically enriched TD. In this section, we describe the approach to generate semantically enriched W3C WoT TDs from the configured iotschema nodes and show how these machine-interpretable descriptions can be stored in a knowledge graph for discovery. This is step 2 in our approach as shown in Fig. 1

6.1. *Thing description generator node*

We introduce a node in Node-RED called *Thing Description Generator*, that generates a TD with semantic mark-ups from iot.schema.org, using configured iotschema nodes as input. The algorithm for generating a TD from SHACL shapes is described in detail in our previous publication [33]. In this paper, we describe the procedure to generate TDs in Node-RED using the *Thing Description Generator* node.

Let us assume that a machine builder wants to generate a semantically enriched TD for his thing. Then he performs the following steps: he searches for the required iotschema interaction pattern nodes, configures the semantics of them as described in Sections 4 and 5; then he wires the output of iotschema nodes (first output which is a SHACL shape) to the input of the *Thing Description Generator* node as shown in Fig. 6. The *Thing Description Generator* takes the semantically configured interaction patterns of a thing (that is configured iotschema nodes) as input and then it constructs a thing description with the interaction patterns. In addition, the *Thing Description Generator* node itself can be configured to define the endpoint where the thing can be accessed, name, id of the thing etc. Thus the output of the *Thing Description Generator* is a semantically enriched TD, that can be stored in a knowledge store.

The TD of the thing can then be stored in a knowledge graph as shown in Fig. 3, for example in the *Thing Directory* which is a knowledge graph with RESTful interface provided by W3C WoT community. In order to simplify the process of storing and discovering things TDs in a knowledge graph, we created a node called *Thing Directory* node which acts as a Thing Directory client. Using this node, a user can store his TD in the directory which enables him to discover, query, modify or delete the TD.

The approach can be used to generate TDs for simple things and also for complex automation systems such as the FESTO workstation presented in our use case.[21] Fig. 6 shows the TD generation in Node-RED for all the sensors and actuators deployed on the FESTO workstation. The figure shows that the semantics of all the sensors and actuators on FESTO workstation are configured using corresponding iotschema

[21] https://www.festo-didactic.com/int-en/learning-systems/process-automation/compact-workstation/mps-pa-compact-workstation-with-level,flow-rate,pressure-and-temperature-controlled-systems.htm

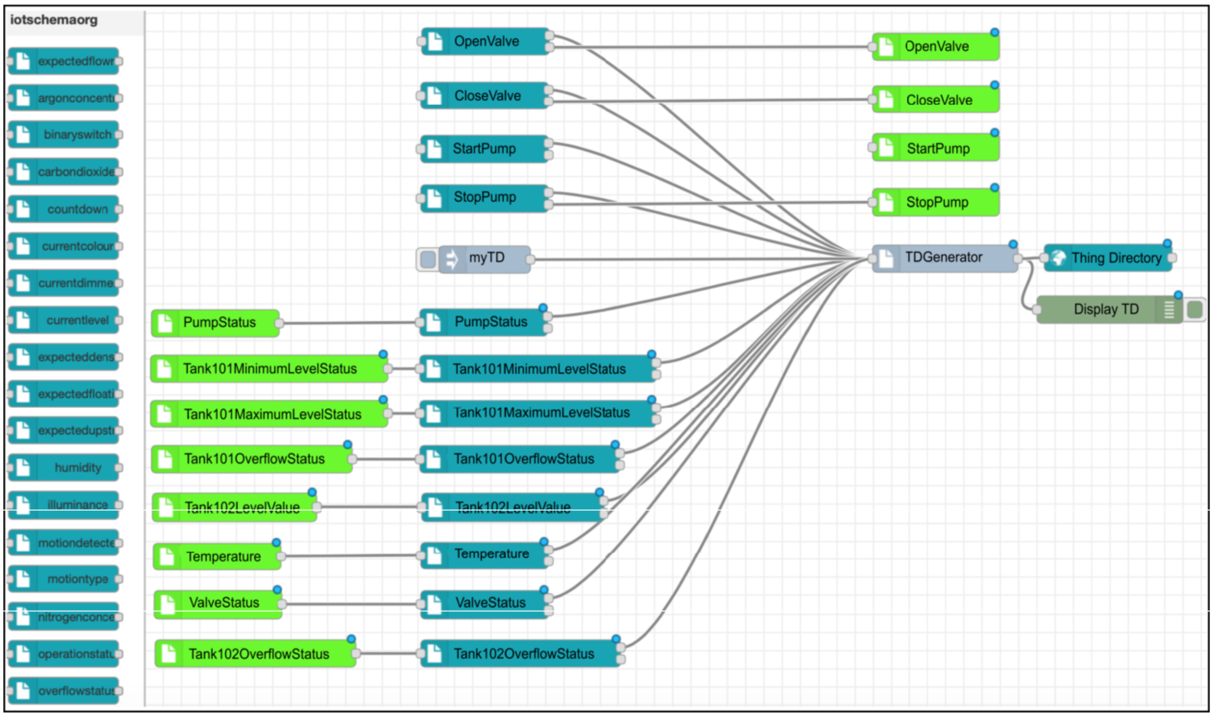


Fig. 6. Usage of semantic Node-RED tool to generate semantically enriched W3C WoT thing description for all the sensors and actuators on FESTO workstation. The green color nodes represent the device nodes that act as endpoints to access an interaction pattern data on a thing. Petrol color nodes represent the iotschema nodes. The grey node, TDGenerator is the thing description generator node.

nodes. Then the output of the iotschema nodes is given as input to the TD Generator node, that generates the semantically enriched TD of the FESTO workstation. A snippet of the generated TD is presented in Listing 4 (where the TD generation algorithm converts SHACL constraints on the data schema into JSON schema constraints, as TD specifies data constraints using JSON Schema). The output of the TD Generator node is given as input to the Thing Directory node to store the generated TD in the Thing Directory. Then the things are available for discovery and application development.

7. Semantically interoperable IoT application development

For rapid IoT application development, we developed an approach using machine interpretable application templates called *Recipes*, as mentioned in the introduction. In this section, we describe the application development approach in detail and explain how the semantic-driven approach automates the complex application development tasks and how the approach is integrated into Node-RED, these are steps 3 and 4 of our approach as shown in Fig. 1. Using our approach, a user who is not an expert in semantic technologies and an engineer not familiar with the domain can quickly and easily design and develop semantically interoperable IoT applications as Node-RED flows, thanks to the semantic querying which automate the complex application development tasks.

The basis for this work is the concept of Recipes that we published previously in [34] by considering a smart city use case. In that paper we presented the Recipe model and a prototypical implementation of a UI tool to design and develop Recipes. In the current paper, we describe how the Recipe concept is introduced in a widely used IoT application development tool, Node-RED.

A Recipe describes the requirements of a WoT application in terms of *Ingredients* and *Interactions*. An Ingredient represents an interaction pattern of a thing with certain capabilities required for an application. Interactions specify the data-flow between Ingredients in order to execute the application. In Node-RED, an iotschema node is treated as a Recipe Ingredient, since an iotschema node acts as a machine-interpretable

```
{
  "@type": [
    "Thing",
    "iot:Pump",
    "iot:Valve",
    "iot:FloatSwitch",
    "iot:UltrasonicSensing",
    "iot:ProximitySensing"  ],
  "id": "urn:dev:wot:siemens:festolive",
  "name": "FestoLive",
  "iotcs:deviceModel": "urn:com:siemens:wot:festo",
  "security": [{ "scheme": "basic" }],
  "properties": {
    "PumpStatus": {
      "@type": "iot:OperationStatus",
      "iot:capability": "iot:Pump",
      "iot:isPropertyOf":{"@id":"Pipe2", "@type":"iot:LiquidPipe"},
      "type": "object",
      "properties": {   "PumpStatus": { "type": "boolean" } },
      "writable": false,
      "observable": false,
      "forms": [
        {
          "href": "coap://192.168.0.101:5683/PumpP101/status",
          "mediaType": "application/json"
        }
      ]
    },
    "ValveStatus": {
      "@type": "iot:OperationStatus",
      "iot:capability": "iot:Valve",
      "iot:isPropertyOf":{"@id": "Pipe1", "@type":"iot:LiquidPipe"},
      "type": "object",
      "properties": {   "ValveStatus": { "type": "boolean" } },
      "writable": false,
      "observable": false,
      "forms": [
        {
          "href": "coap://192.168.0.102:5683/status",
          "mediaType": "application/json"
        }
      ]
    }
   }
  }
```

Listing 4. A snippet of semantically enriched FESTO workstation thing description generated by semantic Node-RED tool

template to describe the capabilities of an interaction pattern of a thing. An iotschema node can thus be used to describe the required things of a Recipe. The dataflow between Ingredients is represented by wiring the Ingredient nodes with each other. In Node-RED, the IoT applications are called *Flows*. Therefore, we call Recipes in Node-RED *Recipe Flows*, that represent a semantically interoperable application template. They can be stored in a knowledge graph such as Thing Directory for discovery and re-use.

As we mentioned earlier that Recipe flows are semantically interoperable IoT application templates.

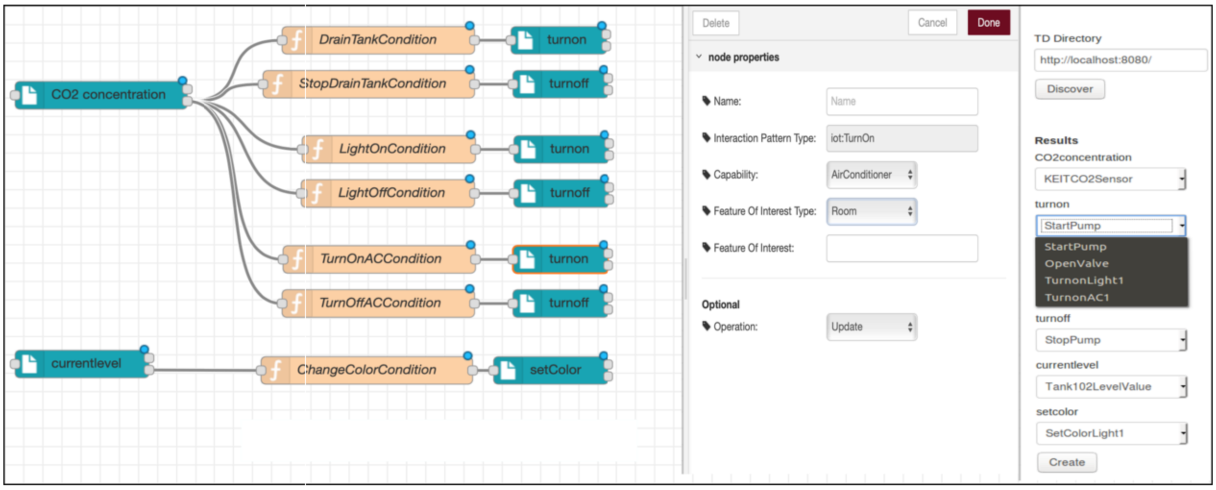
They are not bound to things from any specific platform, ecosystem or domain. This enables a Web developer or IoT application developer to create a *Recipe Flow* without thinking about the existing things. If a user is interested in an application, then he can simply discover a required *Recipe Flow* from Thing Directory. In order to bind the Recipe Flow with his things, our approach provides automated discovery of the matching things. Therefore, one can easily instantiate a Recipe Flow with one's things and deploy the application. Thus, the Recipe Flows are re-usable, they can be instantiated with things from diverse device vendors and platforms. In the following sections we will present in detail the application design and development approach using Recipe Flows.

7.1. Design a Recipe flow

Creating a Recipe flow is as simple as creating a flow in Node-RED. A Web developer or IoT application developer simply drags and drops the Ingredients, that is, required iotschema nodes for his application. He then configures the iotschema nodes to specify the functional requirements of the application. That is, he can assign a capability, feature of interest, data scale, unit of measurement, etc. to an iotschema node to describe the application requirements. Then, he can write an application logic by using *function* nodes in Node-RED. He then describes the data-flow between the ingredients by wiring the nodes into a flow.This corresponds to step 3 of our approach as shown in Fig. 1 For every flow, Node-RED provides a JSON description, which can be saved into the flow library of Node-RED. However, the flow description is not machine-interpretable, therefore, it cannot be easily discovered and semantic reasoning cannot be employed on such a flow description. In order to overcome this problem, we developed a simple semantic model to describe Node-RED Flows and generated a JSON-LD context from it. By adding the context to a JSON flow description, it can be easily enriched with semantics. Thereby, the JSON flow description is represented in the JSON-LD format and is enabled to be stored in a knowledge graph such as Thing Directory. Moreover, semantic querying using SPARQL queries can be used for efficient discovery of Recipe flows and to automate application development tasks such as skill matching. For this purpose, we introduce a new node called the *Matchmaker*.

A user can search the flow library or Thing Directory to discover the required Recipe flow for his application and instantiate it with the matching things on his IAS using the *Matchmaker* node. The functioning of this node is presented in detail in the next sections using the following use case.

Figure 7 shows the Recipe flow designed for our industrial integration use case presented in Section 3. The Recipe flow represents the following application: when the CO_2 concentration in a chemical factory exceeds a certain limit, then drain the liquid from the tank, turn on a flash light to warn the employees in the factory and also turn off the air conditioners in the factory based on the CO_2 concentration. The application also checks for the current liquid level in a tank and



Name: Name
Interaction Pattern Type: iot:TurnOn
Capability: AirConditioner
Feature Of Interest Type: Room
Feature Of Interest:

Optional
Operation: Update

TD Directory
http://localhost:8080/

Results
CO2concentration
KEITCO2Sensor
turnon
StartPump
StartPump
OpenValve
TurnonLight1
TurnonAC1
turnoff
StopPump
currentlevel
Tank102LevelValue
setcolor
SetColorLight1

Fig. 7. Recipe flow designed in semantic Node-RED for the industrial integration use case & discovery of matching things for the Recipe flow by matchmaker node.

changes the color of the light. The interesting point to be observed in this Recipe flow is that the iotschema nodes *turnOn* and *turnOff* are used three times in the Recipe flow and each time they are configured differently to model application requirements. That is, the nodes are configured with different capabilities such as *Pump* (to start the pump to drain liquid from the tank), *Light* (to turn on a light), *Air conditioner* (to turn on an air conditioner) respectively. Figure 7 shows that such a complex application can be created easily by dragging and dropping the required iotschema nodes, configuring them and wiring them together. Then, the user can use the *Matchmaker* node in order to connect to the knowledge graph where his TDs are stored. Then the *Matchmaker* discovers the compatible things on which the *Recipe Flow* can be instantiated. In the following section, we will describe in detail about automated application development using the *Matchmaker* node.

7.2. Automated application development

A Recipe ingredient describes the skills that is capability, interaction pattern, feature of interest and data schema of a thing required for an application. A SPARQL query is generated from an ingredient description. The SPARQL query is then executed on a thing directory or knowledge store where the TDs of a workstation are stored. Then the query gives the matching things on the workstation, that can run the application as a result. Currently the matchmaking process is a binary case. That is only full match cases are considered. The *Matchmaker* node automates matchmaking process. Moreover, it also enables a user to select a thing (from discovery results) for each Ingredient and instantiates the Recipe flow with selected things to create an application or a *Recipe instance flow*. Listing 5 shows the SPARQL query that is generated from the Recipe ingredient node, TurnOn of AirConditioner.

We described the matchmaking process in our previous publication [34] in detail. Here we elaborate on central and distributed discovery approaches for matchmaking. Firstly, semantically enriched TDs of things are stored in a knowledge graph as described in the previous sections. The knowledge graph can be hosted centrally in a Cloud or it can be hosted locally on the Edge of an IAS or BAS, that is, on an Edge device deployed on an IAS or BAS based on the user's requirements. For both Cloud and Edge approach, we describe here two methods for discovery. They are: (1) central discovery and (2) distributed discovery.

7.2.1. Central discovery

If the semantically enriched TDs are stored on the Cloud which hosts a knowledge graph such as Thing Directory, then the central discovery approach is used for skill matching, that is, to discover things that comply to the requirements of a Recipe. Standard semantic querying using SPARQL queries can be applied for discovery in the central approach. The Matchmaker node running on Node-RED connects to the knowledge graph hosted on the Cloud and does the matchmaking process.

Figure 8A shows the system architecture for central discovery. Here we consider a FESTO PA workstation as an IAS. The figure shows that TDs of all the field devices deployed on the FESTO workstation

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX schema: <http://schema.org/>
PREFIX iot: <http://iotschema.org/>
PREFIX td: <http://www.w3.org/ns/td#>

SELECT DISTINCT ?interactionName WHERE { GRAPH ?g  {
    ?t rdf:type  td:Thing .
    ?t td:interaction ?interaction .
    ?interaction td:name ?interactionName .
    ?interaction rdf:type  iot:TurnOn .
    ?interaction iot:capability iot:AirConditioner .
    ?interaction iot:featureOfInterest ?foi .
    ?interaction ?p ?data .
    }
}
```

Listing 5. Complex SPARQL query to discover a TurnOn interaction of a thing of type air conditioner

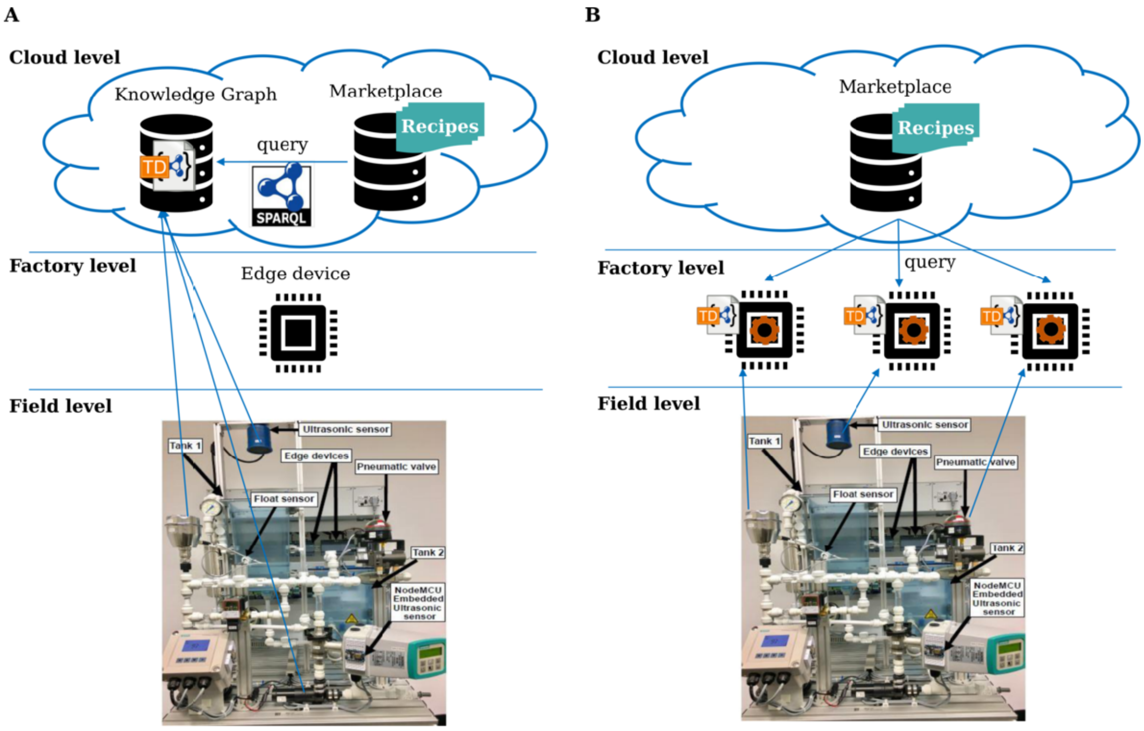


Fig. 8. System architecture: (a) central discovery on the cloud, (b) distributed & local discovery on the edge.

are stored in Thing Directory hosted on the Cloud. Using the Matchmaker node, a user can connect to Thing Directory from Node-RED. The Matchmaker generates queries from the Recipe flow description and executes the generated queries on Thing Directory, in order to discover matching things for a Recipe. Since the Cloud has sufficient resource in terms of computational power and memory, the performance for discovery is good. The central discovery approach for application development is feasible and scalable. We conducted extensive experiments using the central discovery approach; the results are presented in detail in Section 8.

7.2.2. *Distributed discovery*

If the semantically enriched TDs are stored on an Edge device deployed on an AS, then the distributed discovey approach is used to disccover matching things to instantiate a Recipe. Figure 8B shows the system architecture for distributed and local discovery on the Edge of an AS. In this case we again consider a FESTO PA Workstation as the IAS. For Edge processing, one or more Edge devices can be deployed on an IAS or BAS and semantically enriched TDs can be stored on them as shown in Fig. 8B. This is in contrast to the central approach where TDs are stored on the Cloud. In distributed discovery approach, the TDs of the things on an AS are distributed among all the Edge devices deployed on the AS. This way, local intelligence can be achieved by employing semantic querying on the Edge devices. Nevertheless, IoT Edge devices are resource constrained when compared to the resources on a Cloud, thus PC-based semantic querying techniques are not feasible on the Edge devices. Therefore, we propose to use non-standard semantic querying techniques on the Edge devices [7]. We propose here to do querying using datalog on an Edge device. For this purpose, we deployed a column-oriented datalog reasoner called *VLog* [36] on an Edge device.[22] Using VLog, we can store the semantic descriptions of things on an Edge device of an AS. Then we can perform semantic querying and reasoning to discover matching things required to run an application as shown in Fig. 8B. This way, we can support a user in decision making process, to decide whether an application should be deployed on the AS or not.

Depending on the complexity and number of things on an IAS or BAS, one or more Edge devices can be deployed on it. In order to do the discovery on the Edge

[22] https://github.com/karmaresearch/vlog

of an AS, the VLog engine is installed on all the Edge devices and distributed querying techniques are employed. The distributed querying approach on the Edge of an AS is explained in detail in our previous publication [35]. In this paper, the focus is on integrating the distributed querying with Node-RED. The VLog engine on an Edge device can be offered as a Web service, which enables a user to connect to the engine remotely (from Node-RED). It offers flexibility to a user to send his TDs to an Edge device from Node-RED and also to send matchmaking queries easily to the VLog engine on an Edge device. Similar to the central approach, a user can use the Matchmaker node to connect to the VLog engine running on an Edge device, where the distributed discovery takes place. We conducted extensive experiments to evaluate the feasibility, performance and scalability of the local & distributed discovery approach using the VLog engine. The results are presented in Section 8.

7.3. *Deployment of an application*

The user can deploy an application, that is, an instantiated Recipe flow either locally on his computer, on the Cloud, or on an Edge device wherever Node-RED can run.

This is also a novel feature we introduce in Node-RED, since, until today, it is only possible to deploy Node-RED flows locally. In our approach, we extended this feature by offering a flexibility to the user to deploy an application wherever he wants, provided that Node-RED is running on that machine, since the Node-RED run-time is required to run an application. For this purpose, we provided a new node in Node-RED called *App Deployer*. A user can configure this node to give the URI (IP address) where the application should be deployed. Then the application will be sent to Node-RED running on that address and, it will be deployed. Therefore, our approach offers flexibility to the user to deploy an application anywhere. If it is an application where the data from multiple automation systems is collected and processed or an application whose results should be shared with different people then, he can deploy it on the Cloud. If it is an application performing local analytics or orchestrating things locally on an AS, then, he can deploy it on the Edge of an AS such that it works efficiently without high network latency. Otherwise, he can simply run the application locally on his PC.

8. Evaluation

We performed extensive experiments to evaluate our rapid WoT application development approach presented in this paper. The experiments are conducted with a real FESTO Process Automation Workstation shown in Fig. 8 with the real data from the things on it. We conducted four experiments to perform a quantitative and qualitative evaluation of our approach. The aim of the quantitative evaluation is to check the feasibility of the automated application development approach using *Matchmaker* in real world scenarios.

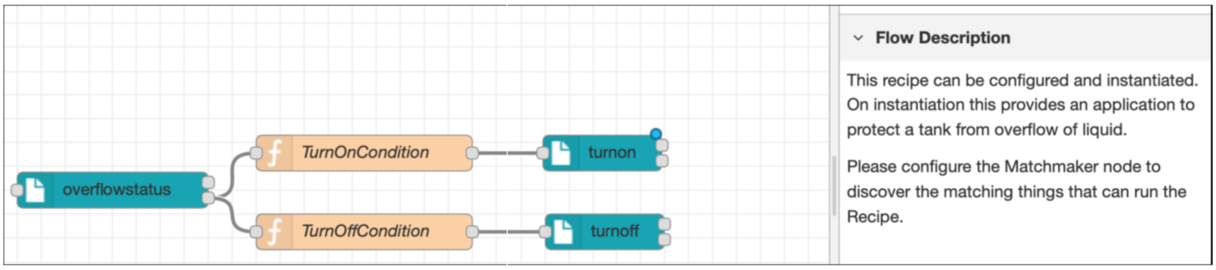
8.1. *Quantitative evaluation*

Two sets of experiments are conducted for the quantitative evaluation. The aim of the first experiment is to analyze if the local discovery approach is feasible to be done on an Edge device. Another goal is to test if the performance of the approach is good and scalable in real life scenarios. The aim of the second experiment is to compare the performance of *Matchmaker* between central discovery and local discovery approaches.

8.1.1. *Experimental setup*

The FESTO PA workstation used in the experiments has eight sensors (two capacitive sensors, two float switches, ultrasound sensor, flow sensor, pressure sensor, temperature sensor) and three actuators (pump, two-way ball valve with pneumatic quarter turn actuator and end-position sensing, heater). Therefore, there are 11 physical things on our IAS with a total of 17 interaction patterns (Properties/Events/Actions) on them. A SIMATIC IOT2000[23] device is deployed on the workstation, which acts as the Edge device on the IAS. SIMATIC IOT2000 is an IoT gateway, which can be deployed on an IAS or BAS. It has an Intel Quark x1000 operating system and 512 MB RAM. The datalog engine, VLog is installed on it. With this setup, we created semantically enriched TDs for all the things (field devices) on FESTO as shown in the Fig. 6. For the evaluation of the local discovery approach the TDs is stored in the VLog engine installed on IOT2000. For evaluation of the central discovery approach, the TDs are stored in *ThingDirectory* running on a PC with 512 GB storage and 4 GB RAM. In this experiments, the PC plays the role of the Cloud.

[23] https://www.siemens.com/global/en/home/products/automation/pc-based/iot-gateways/iot2000.html



This recipe can be configured and instantiated. On instantiation this provides an application to protect a tank from overflow of liquid.

Please configure the Matchmaker node to discover the matching things that can run the Recipe.

Fig. 9. Tank overflow protection Recipe.

Table 1
Experimental set up on FESTO PA workstation for quantitative evaluation

Dataset	Number of things	Number of datapoints
1	11	17
2	55	85
3	110	170
4	165	255
5	220	340

For the experiments, we considered the overflow protection Recipe flow shown in Fig. 9. For this Recipe Flow, the Matchmaker generated three queries to discover matching things for the Ingredients of the Recipe. With this data, we conducted experiments on the Edge and the PC to check time taken to load TDs into VLog and time taken by VLog to materialize the data with the given set of datalog rules. We first conducted the experiment with the original data set that is a TD with 11 sensors & actuators. Then we linearly increased the data set by a factor of five with simulated data. That is, we conducted experiments with the data set for 11 things, 55 things, 110 things, 165 things and 220 things as shown in Table 1. With this setup, we conducted the experiments. The results of the experiments are presented in the following section.

8.1.2. *Results*

For the local discovery evaluation, each data set shown in Table 1 was stored on the SIMATIC IOT2000 device deployed on the FESTO workstation. We checked the time taken to load and materialize the data in the VLog engine for each data set. The results are presented in Fig. 10A. The figure shows that the time taken to load the TD data for 11 things is 7.2 seconds and the time taken to load TD data for 220 things is 10.8 seconds. This is due to the fact that the Edge device under our consideration is resource constrained in terms of memory and processing power. Moreover, in real life scenarios a AS is equipped with more than one Edge devices and each Edge device is connected to a maximum of 20 things. Therefore, the resulting time to load data in the VLog engine is acceptable as the results occur during engineering or application development time but not at run-time.

The second part of the experiment was to test the time taken to execute the queries for each data set on the VLog engine and the Thing Directory running on the PC. The results are presented in Fig. 10B. The graph shows that the VLog engine takes 4.5 seconds for query execution on a data set for 220 things whereas, the Thing Directory running on the PC takes 2.2 seconds for the same query and the same data set. Therefore, we clearly see that the central discovery approach on the PC performs better than the local discovery on the Edge. This is because the PC has more resources than are available on a constrained Edge device. Nevertheless, the results are acceptable, since it is during application development time but not run-time. Moreover, as we argued before, typically 20 things are connected to an Edge device in real life scenarios. Therefore, we can conclude from our experiments that it is feasible to perform automated development of applications on the Edge device deployed on an AS. Furthermore, the approach is scalable. The performance can be improved further by using more powerful reasoning engines[24] and by using more powerful Edge devices such as SIMATIC Nanobox,[25] Raspberry Pi,[26] etc with more storage and computational power.

8.2. *Qualitative evaluation*

The aim of the qualitative evaluation is to analyze the usability and applicability of the semantic Node-

[24] https://cecs.anu.edu.au/events/souffle-datalog-engine-static-analysis

[25] https://w3.siemens.com/mcms/pc-based-automation/en/industrial-pc/box-pc/simatic-ipc227e/pages/default.aspx

[26] https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

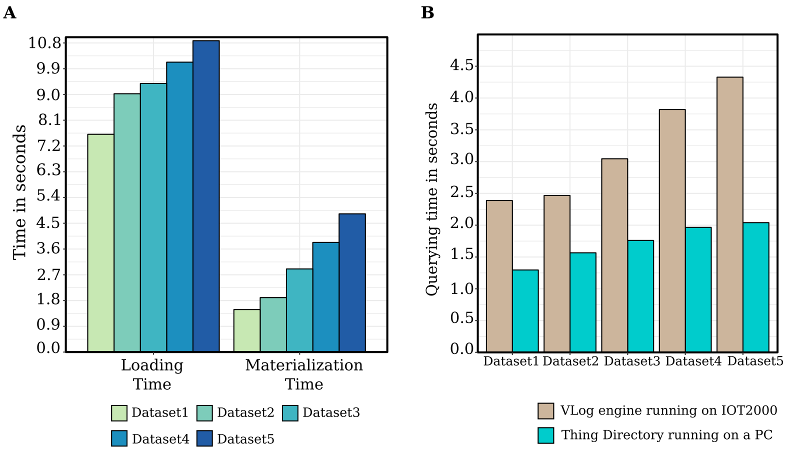


Fig. 10. Results of quantitative evaluation on FESTO PA workstation (a) the bar graph represents the results of loading time and materialization time for each dataset on SIMATIC IOT2000 edge device for local discovery (b) the bar graph represents the results of comparison between querying time in local and central discovery approaches.

RED tool for interoperable WoT application development. The Node-RED tool extended with semantic nodes is a very novel approach and provides an end to end solution for WoT application development. In order to evaluate the features of this tool, we conducted an experiment to compare the features of our tool with existing tools for WoT application development. We also compared our tool with the tools available to use semantic models and the tools to do engineering of new applications on an IAS. Our semantic Node-RED tool is used to design and develop following classes of applications: non-time critical applications, diagnosis applications, applications at the level of MES (Manufacturing Execution System), SCADA (Supervisory Control and Data Acquisition), etc. Therefore we choose the tools that are used to create these classes of applications.

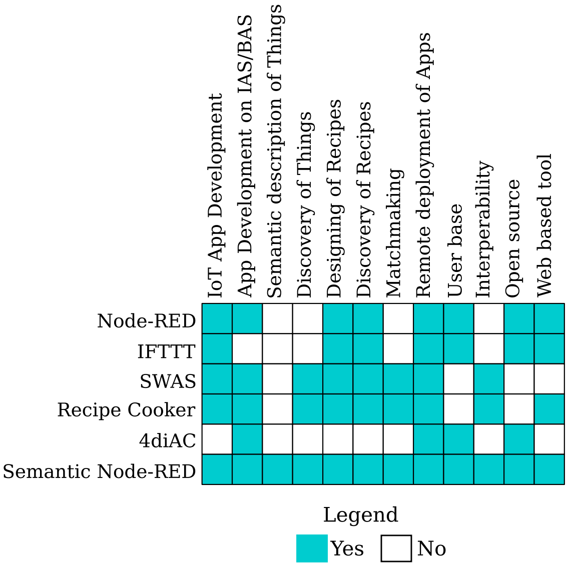
We conducted a second set of experiments to analyze the usability of the tool by users who are semantic and domain experts for developing WoT applications. The tool was demonstrated to over 100 experts including WoT experts during the W3C WoT PlugFest,[27] to industry experts during Global University Challenge for Automation Meets Edge[28] and to Siemens engineers during user evaluation. In the following sections, we detail the experiments conducted and, the results obtained. We present the users feedback about pros and cons of the tool, the scope for improvement, limitations of the tool and future directions for the development of the tool.

8.2.1. Feature comparison

There are few tools available for engineering applications on complex IAS such as 4DiAC. On the other hand, there are several tools available for IoT application development such as Node-RED, IFTTT [29], IoTivity,[29] glue.things [17], 3TwoOne,[30] etc. There is a tool available for WoT application development called WoTKit [3]. However, semantics is not part of any of the above mentioned tools. Few tools exist for semantic-based IoT application development such as ReApp,[31] AllJoyn [38], SWAS [35] (Semantic Web of Things for Automation Systems), or RecipeCooker [34]. These tools integrate semantics to be used in their Integrated Development Environment (IDE) for marking-up IoT sensors/actuators, robots with semantics, for discovery of things, to display semantic models graphically, etc. All these tools constitute the related work for our work on extending Node-RED with semantics.

We evaluated the features of some of the above mentioned tools against the requirements of a semantic-driven tool for rapid application development on complex ASs. We evaluated Node-RED, IFTTT and 4DiAC as they are open source. We also evaluated SWAS and RecipeCooker as these tools were accessible for us. We could not evaluate glue.things,

[27] https://github.com/w3c/wot/tree/master/plugfest/2018-lyon
[28] http://www.siemens.com/automation-meets-edge-challenge
[29] https://www.iotivity.org/
[30] https://3twoone.com/
[31] http://www.reapp-projekt.de/



	IoT App Development	App Development on IAS/BAS	Semantic description of Things	Discovery of Things	Designing of Recipes	Discovery of Recipes	Matchmaking	Remote deployment of Apps	User base	Interperability	Open source	Web based tool
Node-RED	■	■	□	□	■	■	□	■	■	□	■	■
IFTTT	■	□	□	□	■	■	□	■	■	□	■	■
SWAS	■	■	□	■	■	■	■	■	□	■	□	□
Recipe Cooker	■	■	□	■	■	■	■	■	□	■	□	■
4diAC	□	■	□	□	□	□	□	■	■	□	■	□
Semantic Node-RED	■	■	■	■	■	■	■	■	■	■	■	■

Legend

■ Yes □ No

Fig. 11. Heatmap representing the results of the feature comparison between various industrial engineering and application development tools.

3TwoOne, IoTivity, ReApp and WoTKit in detail, since they are not publicly available. In this section, we will provide the evaluation results in detail. The Heatmap shown in Fig. 11 presents the feature comparison results between these tools.

The features for comparison are chosen with focus on the requirements for a semantic-driven IoT application development tool, a tool which can create an ecosystem for device vendors, machine builders, IoT application developers, AS engineers and Web developers to easily use semantic models for purposes such as: (1) semantic mark-up of simple and complex things; (2) designing, developing, discovering, sharing, reusing and extending WoT applications on complex IAS or BAS. Based on these requirements we derived the following features for comparison.

(1) Is the tool suitable for WoT application development?
(2) Is the tool suitable for application development on complex IAS?
(3) Can a user describe a thing affordances semantically using the tool?
(4) Does the tool support discovery of required things?
(5) Can a user create application templates using the tool?
(6) Does the tool support discovery of application templates?
(7) Can matchmaking be done using the tool, which enables rapid application development?
(8) Does the tool support the applications to be deployed remotely?
(9) Is there a user community for the tool to create and share the applications?
(10) Does the tool enable IoT semantic interoperability?
(11) Is it open source?
(12) Is the tool Web friendly?

The evaluation of the tools is carried out based on these questions. The results of the evaluation are shown in Fig. 11 and we present a detailed discussion of each tool below.

Node-RED: Node-RED in the current state is a Web friendly tool to wire IoT hardware and software components. It is a low-code platform that enables a user to easily design and develop applications without focusing on writing the code for an application. However, as mentioned earlier, interoperable WoT applications cannot be developed on Node-RED, since semantics is not a part of Node-RED in the current state. Moreover, applications should be deployed locally in Node-RED. The features of current Node-RED are discussed in detail through out the paper.

IFTTT: IFTTT is a Web-based tool for very quick and easy IoT application development using applets. It provides a user-friendly interface to discover and instantiate applets. The applets in IFTTT are very easily usable and highly reusable. However, there is only one fixed pattern in IFTTT to create applications:"If this, then that". If a user wants to create a complex pattern, then, he has to combine many applets which is not user friendly. Further, IFTTT is suitable for simple IoT applications, but it is not suitable for creating applications for complex ASs. Moreover, the applets are bound to a specific device ecosystem (for example, some applets work only with the Amazon Alexa device, Philips Hue devices etc.) and they cannot be used with devices from other ecosystems. Semantic service discovery or matchmaking are not features of IFTTT. IFTTT does not enable users to create interfaces for their devices or to provide semantic description for their devices.

SWAS tool: The advantages of the SWAS tool [35] are that the tool is suitable for developing semantically interoperable WoT applications on complex IAS and semantics is integrated in the SWAS tool. It has features such as semantic discovery of things, design and discovery of Recipes, matchmaking and remote deployment. However, semantic description of a thing is not a feature of the SWAS tool. Moreover, it is an eclipse-based tool, which makes it heavy weight and less Web friendly. The SWAS tool is implemented in Java which limits its deployment on smart devices or IoT Edge devices. Moreover, the tool is not open source.

Recipe-cooker: The Recipe-cooker tool [34] is a browser based editor for creating semantic application templates for building applications for automation, smart cities, etc. The tool is implemented in Node.js, therefore, it is lightweight and Web-friendly. A user can design semantic Recipes, discover them, do matchmaking and instantiate Recipes using this tool. However, this tool does not support semantic description of a thing. A user has to use another tool to create semantic description of an IoT offering. Moreover, the tool is not open source and it does not have a strong user base. Therefore, there is very limited community support.

4DiAC: 4DiAC is an eclipse based engineering tool for engineering applications on complex IAS or BAS. The tool enables a user to efficiently develop Function Blocks, test them and deploy them on a PLC or Raspberry Pi or PC. However, the tool is not meant to be used for IoT application development and semantics is not integrated into the tool.

Semantic Node-RED: The tool was created to meet all the requirements mentioned above. It is an extension of the widely used Node-RED tool. Instead of developing a new tool we extended Node-RED which is open-source and has a huge user base. All the nodes, flows and extensions that we developed for Node-RED for semantic extension are open source. We believe that this extension will enable a perfect ecosystem for semantic-driven rapid application development on complex ASs. We envision that, using this tool, we enable device vendors not only to create semantic descriptions for their things, but also to provide semantically interoperable application templates that can integrate physical devices across diverse platforms and ecosystems.

8.2.2. User evaluation

Our basic assumption for the user evaluation is that a user has some basic knowledge about the things on the AS. In our use case, the assumption is that the users participating in the evaluation have the knowledge about the basic functionality of a valve, pump, etc. **Automation Meets Edge Challenge**: The Semantic Node-RED tool was first exhibited at the Global University Challenge for Automation meets Edge competition conducted by Siemens at Nuremberg, Germany, in October, 2018.[32] It was a world wide competition, in which students from 55 universities participated. At this competition, we presented our idea entitled "Rapid IoT application development approach" and presented the semantic Node-RED tool to design and develop IoT applications. The applications developed are then deployed on the Siemens new industrial Edge device called SIMATIC Nanobox. In this competition, we won the "Highest Business Impact" award for our idea. Several industrial experts and students participated in this event. It gave us a good opportunity to present the tool to industrial experts and get their feedback about our approach and the tool. In the following we summarize the feedback.

22 experts participated in the evaluation. They expressed that the approach:

- addresses key issues in IoT such as the need to use common and structured data models and need for simplified application development.
- simplifies the task of data users to structure complex data using common data models.
- enables anyone to become an application developer to develop simple applications on automation systems quickly and easily.
- encourages creating different applications using common & well-defined semantic models.
- saves the costs and brings added value.
- creating the application on the Cloud and optionally deploying it on the Edge or the Cloud offers flexibility to the users.

On the other hand, they noted that it would be difficult to have a data model that is common to all the suppliers, and that the applications might be limited for real world scenarios. They expressed that, there should be the possibility to deploy the applications first in a simulated environment to test them before deploying them on the real equipment.

W3C WoT Face to Face Meeting, Lyon: A W3C Face to Face meeting was conducted in Lyon, France in October, 2018. A Plugfest was conducted on October 20–21 as part of the Face to Face meeting.[33] WoT experts from various companies such as Intel, Siemens, Fujitsu, Panasonic, Oracle, SmartThings, etc. participated in this event with their physical devices and WoT

[32] http://www.siemens.com/automation-meets-edge-challenge

[33] https://github.com/w3c/wot/tree/master/plugfest/2018-lyon

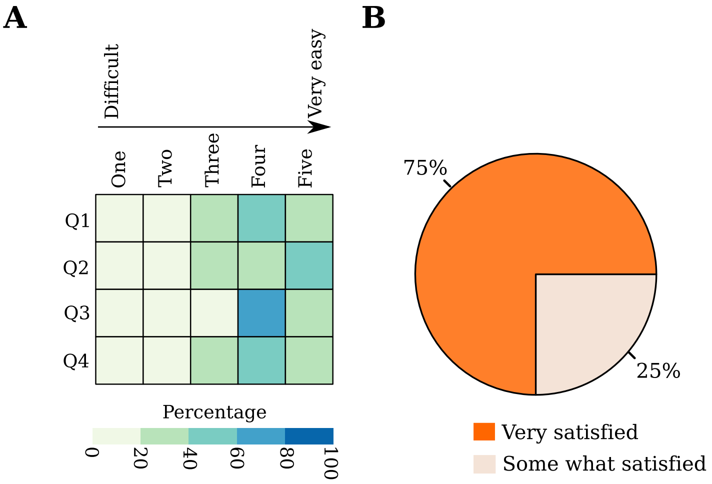


Fig. 12. Results of the user evaluation. The heatmap (a) represents the evaluation results for questions 1 to 4, the pie chart (b) represents the evaluation results for question 5.

demos. On this occasion, we presented the semantic Node-RED tool to the Plugfest participants demonstrating how to create semantically enriched TDs for the physical devices from the participants using the tool. Using iotschema nodes on Node-RED, we were able to create several Recipe flows very easily in few minutes. With the Recipe flows we demonstrated semantic interoperability and how devices from different device vendors can be used to instantiate a Recipe flow in order to create WoT applications. A Recipe to flash a warning light when the oxygen level is under a certain threshold, and a Recipe to protect a tank from overflow of liquid are just a few examples of Recipe flows demonstrated at the Plugfest. We collected informal feedback from the Plugfest participants. The overall feedback about the approach to rapidly develop WoT applications using Recipe flows in Semantic Node-RED tool was very good. We were able to create WoT applications within few minutes using the approach.

User Evaluation with Siemens Engineers: In order to evaluate the usability of the tool for modeling a thing's affordances and data schema with IoT semantics, and to design and develop semantically interoperable applications we conducted a user evaluation of the tool with Siemens engineers (who are not experts in semantics). We created a questionnaire[34] to collect their feedback. The engineers were first given short introduction to the tool. Then they were given two tasks: (1) develop a semantic description of a thing on the FESTO workstation using the tool; (2) develop an application on FESTO: to do this, the user should design a Recipe flow for an application on FESTO, use the *Matchmaker* node to discover compatible things on FESTO, instantiate the application with discovered things and deploy it on the workstation. The engineers were able to do the tasks in approximately 15 minutes. Then we collected their feedback using the questionnaire. Nine engineers participated in the evaluation and we got good feedback from them. We present the evaluation results in Fig. 12.

We asked the following five questions to the users:

Q1. Do you find it easy to use iot.schema.org nodes in Node-RED in order to enrich a Thing Description (when compared to common practice i.e., finding required semantic description from the iot.schema.org website and editing a Thing Description manually)?
Q2. Do you find it easy to use semantic models from Node-RED when compared to existing approaches such as schema.org, SSN Ontology, SAREF Ontology etc.?
Q3. Do you find it easy to work with iot.schema.org nodes?
Q4. Do you find it easy to create WoT applications as flows that contain iot.schema.org nodes?
Q5. How satisfied are you with this tool's ease of use?

For the questions 1 to 4, the users answered on a scale of 1 to 5 (1-Difficult, 5- Very easy) rating the usability of the tool for various purposes. The results for the evaluation of these questions is presented in Fig. 12A.

[34] https://goo.gl/forms/wbFitAeUin26Cnp82

The results show that the tool is fairly easy to use by users who are not semantic experts, to work with the semantic models, to enrich a TD with semantics, and to create semantically interoperable IoT applications. The overall satisfaction of the users about the tool is evaluated in Question 5. The results of Question 5 are presented in Fig. 12B, which show that most of the users are very satisfied with the tool.

The users also gave us feedback about some limitations of the tool. For instance, currently the tool does not facilitate to discover required iotschema nodes to create a TD. This is a subject of future work. In future releases, we plan to enable discovery of iotschema nodes in the tool. Additionally, users mentioned that there should be a way to simulate the applications before deploying them on a running AS. However, this is out of scope of this work.

In summery, we conducted a systematic and extensive quantitative and qualitative evaluation of our approach to evaluate the feasibility, scalability and limitations of the tool. The evaluation results are very encouraging and they also pointed us to future directions to enhance our approach.

9. Case study on asset administration shell

Asset Administration Shell (AAS) is one of the key concepts in Industry 4.0 that is used to describe an asset electronically in a standardized manner. The idea of AAS is to structure the information and functions in the context of the manufacturing industry based on the assets in a uniform manner. We conducted an experiment to implement the usage view of the AAS with W3C WoT technologies, since WoT TD provides a platform-independent and format-neutral vocabulary to describe IoT things and the protocol binding provides accesses to things across diverse protocols and platforms seamlessly. Every asset corresponds to a WoT thing. Each asset service is an interaction pattern (property/event/action) of a thing. TD of a thing corresponds to the asset service registry. The knowledge store, Thing Directory where the TDs are stored corresponds to the computing infrastructure of the AAS. With this setup, plug and play use case of AAS is implemented with Recipes using semantic Node-RED tool.

Our experiment showed that AAS can be efficiently implemented with W3C WoT technologies since they are standardized and light-weight technologies. On top of WoT technologies, Recipes and semantic Node-RED tool is very nice addition to implement plug and play use case, since, semantic Node-RED tool enables very simple usage of Recipe and automated matchmaking concepts. Together with these technologies, plug and play can be implemented easily as Node-RED flows. For example, once an application is instantiated and deployed with Recipes, a new thing that is added in a manufacturing unit can be automatically plugged to the application with automated matchmaking. Moreover, a malfunctioning thing used in an application can be replaced automatically with another compatible thing in a manufacturing unit using automated matchmaking. Therefore, AAS ecosystem can use Recipes, matchmaker and semantic Node-RED tool to build AAS applications.

10. Industry challenges faced & lessons learned

Throughout this work we learned that SW technologies are good to apply in Industry 4.0 for various purposes. Industrial automation systems are very complex with large number of machines that are producing huge amounts of data. Usage of SW technologies in industry addresses several issues such as semantic interoperability between systems, querying the machine models and live data from the machines using standard SW technologies and tools. SW also provides standardized technologies for validation such as SHACL, ShEX etc. These technologies can be used to validate the data that is being exchanged between industrial machines and to validate the machine models. However, it is very challenging to introduce new changes in industrial automation. Moreover, it is also challenging to gain community support to push the development of semantic models e.g., iotschema.org.

11. Conclusions & future work

In summary, we investigated the possibility of using Semantic Web technologies and technologies being standardized in W3C WoT, to achieve semantic interoperability in IoT and enable rapid development of interoperable IoT applications. We chose the novel IoT orchestration tool called Node-RED for this purpose. We extended the Node-RED tool with semantic definitions developed by iot.schema.org and we showed how our approach simplifies development of semantically enriched TDs and of semantically interoperable IoT applications. We conducted an extensive qualitative and quantitative evaluation of our approach with real world use cases. The results show that the approach is feasible and scalable in real world scenarios, and that the tool enables engineers who are not domain experts and non-experts in semantic technologies to easily design and develop semantically interoperable WoT applications.

We showed that our approach accelerates application development in many ways: firstly by using low-code platforms such as Node-RED, where lot of functionalities come from the Node-RED ecosystem itself e.g. nodes such as CoAP, HTTP etc. Moreover, we introduced application templates with Recipes which are reusable. Therefore, there is no need to develop an application from scratch.

The current limitations of the tool are as follows: complex applications involving multiple sensors and actuators cannot be created with the tool. Since the tool is currently hosted on AWS, it does not support multi-tenancy. The proposed approach of IoT application development with Recipes is implemented with Node-RED. Node-RED is a good tool for prototyping IoT applications quickly and easily without much domain know-how. However it has limitations such as scalability, performance. Node-RED is not suitable for developing and deploying large number of IoT applications on multiple devices simultaneously, therefore, it is not scalable. Node-RED is only one way of implementation of Recipe technology, however, other tools can also implement the Recipe mechanism.

In the future, we plan to extend the tool with existing industry standards such as the OPC UA semantics. This will facilitate the integration of state of the art automation systems into IoT and also enable rapid application development on them. We will extend Recipes to create complex Recipes with multiple sensors and actuators. Provide semantic templates for machine vendors to describe the semantics of their things and offer them to their customers. Another research direction is to implement the tool we presented in this paper with other low-code development platforms e.g., Mendix.[35]

References

[1] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal et al., Brick: Towards a unified metadata schema for buildings, in: *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, ACM, 2016, pp. 41–50. doi:10.1145/2993422.2993577.

[2] T. Bangemann, S. Karnouskos, R. Camp, O. Carlsson, M. Riedl, S. McLeod, R. Harrison, A.W. Colombo and P. Stluka, State of the art in industrial automation, in: *Industrial Cloud-Based Cyber-Physical Systems*, Springer, 2014, pp. 23–47. doi:10.1007/978-3-319-05624-1_2.

[3] M. Blackstock and R. Lea, IoT mashups with the WoTKit, in: *Internet of Things (IOT), 2012 3rd International Conference on the*, IEEE, 2012, pp. 159–166. doi:10.1109/IOT.2012.6402318.

[4] D. Brickley, Basic geo (WGS84 Lat/long) vocabulary, W3C Semantic Web Interest Group, 2006.

[5] V. Charpenay, S. Käbisch and H. Kosch, Introducing thing descriptions and interactions: An ontology for the web of things, in: *Proceedings of the 1st Workshop on SemanticWeb Technologies for the Internet of Things (SWIT) at ISWC*, 2016, pp. 55–66.

[6] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana et al., Web services description language (WSDL) 1.1, W3C, 2001.

[7] S. Christian and S. René, Rule-based OWL reasoning for specific embedded devices, in: *Proceedings of the 10th International Semantic Web Conference (ISWC'11)*, Springer, Berlin Heidelberg, 2011, pp. 237–252. doi:10.1007/978-3-642-25093-4_16.

[8] M. Compton, P. Barnaghi, L. Bermudez, R. Garcıa-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W.D. Kelsey, D.L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth and K. Taylor, The SSN ontology of the W3C semantic sensor network incubator group, *Web Semantics: Science, Services and Agents on the World Wide Web* **17** (2012), 25–32, ISSN 1570-8268, http://www.websemanticsjournal.org/index.php/ps/article/view/312. doi:10.1016/j.websem.2012.05.003.

[9] W.W.W. Consortium et al., 2014, JSON-LD 1.0: A JSON-based serialization for linked data.

[10] I. Grangel-González, L. Halilaj, G. Coskun, S. Auer, D. Collarana and M. Hoffmeister, Towards a semantic administrative shell for industry 4.0 components, CoRR abs/1601.01556 (2016), 230–237. doi:10.1109/ICSC.2016.58.

[11] A. Gyrard, C. Bonnet and K. Boudaoud, Enrich machine-to-machine data with semantic web technologies for cross-domain applications, in: *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, IEEE, 2014, pp. 559–564. doi:10.1109/WF-IoT.2014.6803229.

[12] M. Hepp, eClassOWL: A fully-fledged products and services ontology in OWL, in: *Poster Proceedings of the 4th International Semantic Web Conference (ISWC2005)*, 2005.

[13] R. Hodgson, P.J. Keller, J. Hodges and J. Spivak, UDT-quantities, units, dimensions and data types ontologies, USA, Available from: http://qudt.org [March 2014] **156** (2014).

[14] F. Jammes and H. Smit, Service-oriented paradigms in industrial automation, *IEEE Transactions on industrial informatics* **1**(1) (2005), 62–70. doi:10.1109/TII.2005.844419.

[15] S. Jeschke, C. Brecher, T. Meisen, D. Özdemir and T. Eschert, Industrial Internet of things and cyber manufacturing systems, in: *Industrial Internet of Things*, Springer, 2017, pp. 3–19. doi:10.1007/978-3-319-42559-7_1.

[35] https://www.mendix.com

[16] E. Kharlamov, B.C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin and I. Horrocks, Capturing industrial information models with ontologies and constraints, in: *The Semantic Web – ISWC 2016–15th International Semantic Web Conference, Proceedings, Part II*, Kobe, Japan, October 17–21, 2016, 2016, pp. 325–343. doi:10.1007/978-3-319-46547-0_30.

[17] R. Kleinfeld, S. Steglich, L. Radziwonowicz and C. Doukas, glue. things: A mashup platform for wiring the Internet of things with the Internet of services, in: *Proceedings of the 5th International Workshop on Web of Things*, ACM, 2014, pp. 16–21. doi:10.13140/2.1.3039.9049.

[18] H. Knublauch and D. Kontokostas, Shapes constraint language (SHACL). W3C Candidate Recommendation, 11 (8)(2017).

[19] J. Kopeckỳ, K. Gomadam and T. Vitvar, hrests: An html microformat for describing restful web services, in: *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, Vol. 1, IEEE, 2008, pp. 619–625. doi:10.1109/WIIAT.2008.379.

[20] J. Kopeckỳ, T. Vitvar, C. Bournez and J. Farrell, Sawsdl: Semantic annotations for wsdl and xml schema, *IEEE Internet Computing* **11**(6) (2007), 60–67. doi:10.1109/MIC.2007.134.

[21] K. Kotis and A. Katasonov, An ontology for the automated deployment of applications in heterogeneous IoT environments, *Semantic Web Journal (SWJ)* (2012).

[22] R. Lara, D. Roman, A. Polleres and D. Fensel, A conceptual comparison of WSMO and OWL-S, in: *Web Services*, Springer, 2004, pp. 254–269. doi:10.1007/978-3-540-30209-4_19.

[23] W. Mahnke and S.-H. Leitner, *OPC Unified Architecture*, Springer Science & Business Media, 2009, pp. 1–17. doi:10.1007/978-3-540-68899-0.

[24] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D.L. Mcguinness, E. Sirin and N. Srinivasan, Bringing semantics to web services with OWL-S, *World Wide Web* **10**(3) (2007), 243–277. doi:10.1007/s11280-007-0033-x.

[25] H. Merz, T. Hansemann and C. Hübner, *Building Automation: Communication Systems with EIB/KNX, LON and BACnet*, Springer Science & Business Media, 2009. doi:10.1007/978-3-319-73223-7.

[26] J. Mineraud, O. Mazhelis, X. Su and S. Tarkoma, A gap analysis of Internet-of-things platforms, *Computer Communications* **89** (2016), 5–16. doi:10.1016/j.comcom.2016.03.015.

[27] I. Modbus, Modbus application protocol specification v1. 1a, North Grafton, Massachusetts (www.modbus.org/specs.php) (2004).

[28] N. O'Leary and D. Conway-Jones, Node-Red a visual tool for wiring the Internet-of-Things, Retrieved July 4, 2017, from http://nodered.org **4** (2017).

[29] S. Ovadia, Automate the Internet with if this then that (IFTTT), *Behavioral & Social Sciences Librarian* **33**(4) (2014), 208–211. doi:10.1080/01639269.2014.964593.

[30] F. Serena, M. Poveda-Villalón and R. García-Castro, Semantic discovery in the web of things, in: *International Conference on Web Engineering*, Springer, 2017, pp. 19–31.

[31] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini and A. Martel, Framework for distributed industrial automation and control (4DIAC), in: *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on Industrial Informatics*, Daejeon, Korea, 13–16 July, 2008, IEEE, 2008, pp. 283–288. doi:10.1109/INDIN.2008.4618110.

[32] M. Thoma, T. Braun, C. Magerkurth and A. Antonescu, Managing things and services with semantics: A survey, in: *Network Operations and Management Symposium (NOMS)*, May 5, 2014, IEEE, 2014, pp. 1–5. doi:10.1109/NOMS.2014.6838366.

[33] A.S. Thuluva, D. Anicic and S. Ruldolph, IoT semantic interoperability with device description shapes, in: *Proceedings of the 15th Extended Semantic Web Conference*, Springer, Crete, Greece, 2018, pp. 409–422. doi:10.1007/978-3-319-98192-5_56.

[34] A.S. Thuluva, A. Bröring, G.P. Medagoda, H. Don, D. Anicic and J. Seeger, Recipes for IoT applications, in: *Proceedings of the Seventh International Conference on the Internet of Things*, Linz, Austria, October, 2017, ACM, 2017, pp. 1–8. doi:10.1145/3131542.3131553.

[35] A.S. Thuluva, K. Dorofeev, M. Wenger, D. Anicic and S. Rudolph, Semantic-based approach for low-effort engineering of automation systems, in: *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, Springer, Rhodes, Greece, 2017, pp. 497–512. doi:10.1007/978-3-319-69459-7_33.

[36] J. Urbani, C.J. Jacobs and M. Krötzsch, Column-oriented datalog materialization for large knowledge graphs, in: *Proc. 30th AAAI Conf. on Artif. Intell.*, 2016, pp. 258–264. doi:10.5555/3015812.3015850.

[37] R. Verborgh, T. Steiner, D. Van Deursen, R. Van de Walle and J.G. Vallés, Efficient runtime service discovery and consumption with hyperlinked RESTdesc, in: *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on*, IEEE, 2011, pp. 373–379. doi:10.1109/NWeSP.2011.6088208.

[38] M. Villari, A. Celesti, M. Fazio and A. Puliafito, Alljoyn lambda: An architecture for the management of smart environments in iot, in: *Smart Computing Workshops (SMARTCOMP Workshops), 2014 International Conference on*, IEEE, 2014, pp. 9–14. doi:10.1109/SMARTCOMP-W.2014.7046676.

[39] W. Wahlster, Chap. Semantic technologies for mass customization, in: *Towards the Internet of Services: The THESEUS Research Program*, Sep 1, 2014, Springer, 2014, pp. 3–13. doi:10.1007/978-3-319-06755-1_1.

[40] A. Wright and H. Andrews, Json schema: A media type for describing json documents, in: IETF, Internet-Draft draft-handrews-json-schema-OO, 2017.