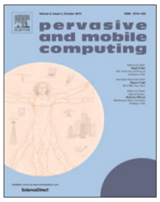
Pervasive and Mobile Computing 87 (2022) 101699

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc



Edge-to-cloud sensing and actuation semantics in the industrial Internet of Things

Marc Vila [a,b,c,*], Víctor Casamayor [c], Schahram Dustdar [c], Ernest Teniente [a]

[a] *Universitat Politecnica de Catalunya, Jordi Girona 1-3, 08034, Barcelona, Spain*
[b] *Worldsensing, Viriat 47, 08014, Barcelona, Spain*
[c] *Distributed Systems Group, TU Wien, Argentinierstrasse 8, 1040, Vienna, Austria*

ARTICLE INFO

Article history:
Received 28 December 2021
Received in revised form 20 August 2022
Accepted 24 September 2022
Available online 29 September 2022

Keywords:
Industrial Internet of Things
Interoperability
Computing Continuum
Context-awareness
Semantics
Autonomous cars

ABSTRACT

There are billions of devices worldwide deployed, connected, and communicating to other systems. Sensors and actuators, which can be stationary or movable devices. These Edge devices are considered part of the Internet of Things (IoT) devices, which can be referred to as a tier of the Computing Continuum paradigm. There are two main concerns at stake in the success of this ecosystem. The interoperability between devices and systems is the first. Mainly, because most of them communicate uniquely and differently from each other, leading to heterogeneous data. The second issue is the lack of decision-making capacity to conduct actuations, such as communicating through different computing tiers based on latency constraints due to a certain measured factor. In this article, we propose an ontology to improve device interoperability in the IoT. In addition, we also explain how to ease data communication between Computing Continuum devices, providing tools to enhance data management and decision-making. A use case is also presented, using the automotive industry, where quickness in maneuver determination is key to avoid accidents. It is exemplified using two Raspberry Pi devices, connected using different networks and choosing the appropriate one depending on context-aware conditions.

1. Introduction

The impact of the Internet of Things (IoT) has had on daily life is undeniable. Continuously expanding; now there are billions of IoT devices connected to the Internet, more than 10 billion, to be precise [1], and this is a growing phenomenon. IoT devices span from simple sensors to consumer and industrial products. From wearables, health trackers, or home appliances to environmental, location, or presence sensors.

One of the main problems we face in the IoT domain is to allow devices to easily communicate data to other devices, since the data format collected by each of them can be very heterogeneous. This problem is known as *Interoperability of Things*, the objective of which is to provide information homogenization so that software systems can provide generic solutions to communicate information in different monitoring environments and application domains. As stated in Elkhodr et al. [2], there is a need to enable interoperability to allow different devices to collect and communicate information with other smart devices or systems.

* Corresponding author at: Universitat Politecnica de Catalunya, Jordi Girona 1-3, 08034, Barcelona, Spain.
E-mail addresses: marc.vila.gomez@upc.edu, mvila@worldsensing.com (M. Vila), v.casamayor@dsg.tuwien.ac.at (V. Casamayor), dustdar@dsg.tuwien.ac.at (S. Dustdar), ernest.teniente@upc.edu (E. Teniente).

https://doi.org/10.1016/j.pmcj.2022.101699

Currently, this communication is possible because there has been several work in standardizing processes, data structures, and manipulation of the data behind these devices. Thus, for instance, Kiljander et al. [3] enable two systems to communicate through the definition of standards and rules. Noura et al. [4] propose a taxonomy to classify the levels of interoperability into *device*, *syntactic*, *networking*, *semantic*, and *platform*. This allows for some kind of communication between IoT devices. However, full interoperability can only be achieved by abstracting the particular syntax and data formats of IoT devices to provide a common semantics for them. The work we propose in this paper contributes to this semantic level by defining an abstract interpretation, that is, an ontology, of the concepts and data being managed in a domain by means of ontologies.

In addition to the *Interoperability of Things*, IoT solutions must also face the problem of deciding where to compute the actuation that must be performed regarding the data of the IoT device. A classical approach is to do it in the *Cloud*, although other computational tiers have been recently developed such as the *Edge* [5,6] or the *Fog* [7]. The main goal of these new proposals is to bring computations closer to the data generation origin, i.e. the device, mainly because of efficiency reasons. Therefore, a new computing paradigm is emerging, the *Computing Continuum* [8], which takes advantage of all computing tiers, from the Cloud to the device level, but also of intermediate tiers such as the Edge or the Fog. In this paradigm, the execution of an application is carried out distributed among all tiers, taking the best of each to fulfill the requirements. Again, the challenge here is to come up with solutions that make this assignment to tiers as transparent as possible so that it does not has to be manually implemented.

The Computing Continuum paradigm allows developing applications in "Smart Industry", "Connected Cars", "Smart Cities", "Logistics", "Smart Energy", or "Healthcare" [8], which will bring about possibilities that are now only in our minds. However, enabling these applications to the complete use of this paradigm together with the Interoperability of Things is still an ongoing issue, even though almost all industrial environments rely heavily on IoT devices. Thus, the IoT and the Computing Continuum paradigms are enabling a new era in the industrial environment, the Industrial IoT. In this new era, devices with sensing or actuation capabilities are connected to low-latency and ultra-reliable communications. Creating a smart, connected, real-time, and collaborative system for efficient monitoring and control of industrial equipment [9].

Nowadays, one of the most challenging domains is that of autonomous vehicles. Providing robust solutions to this domain requires necessarily to address the Interoperability of Things and the Computing Continuum. Autonomous vehicles are full of sensors. Some of them are used for the correct operation of the vehicle, but others serve to collect data on external elements. Information collected must be processed and analyzed quickly to determine whether some action must be taken on the vehicle. Devices will need to communicate with external entities, such as other vehicles, taking into account different delivery priorities. Here is where the Computing Continuum comes into play to decide if the data involve action and reaction at the Edge or if it has to be sent to the Fog or the Cloud. We provide some experiments of our proposal when applied to autonomous vehicles because of the importance of this domain in terms of the problem we face.

The main contributions of this paper can be summarized as follows:

- We propose a semantic ontology for monitoring elements using IoT devices and platforms, from the sensing part to the actuation part; allowing end-users to obtain information, access it, monitor it, and perform actions if necessary. Our ontology contributes to the Interoperability of Things and it is based on existing ontologies such as SSN/SOSA, GeoSPARQL, OWL-Time, and CMTS.
- Our ontology, called IoTMA, is independent of any particular domain, and thus, all software systems dealing with the components of IoTMA will be able to handle all different IoT deployments without having to modify any single line of code. The only component that will have to be developed is how do the different devices provide the ontology with their information.
- We study the implications of linking low-latency response systems and semantic interoperability mechanisms; including state-of-the-art mechanisms to support conceptual generalization of entities; supporting the monitoring of different industrial environments. In this way, the Computing Continuum treatment is also integrated into our ontology.
- We provide an implementation of our approach to a scenario based on autonomous vehicles. This is achieved by using elements external to the vehicle to obtain information about possible dangers on the road and act accordingly through a certain type of communication.

Section 2 reviews the related work. Section 3 describes the main concepts of the ontology we propose here. Section 4 matches the work done with the previous ontologies. Section 5 shows the experimentation carried out for this article. Finally, Section 6 presents our conclusions and points out future work.

2. Related work

We address here related work on the Interoperability of Things, including perspectives from Ontologies and Semantics, and from the Computing Continuum.

2.1. Interoperability of things

Interoperability is the ability of a system to communicate with other systems. A Thing is a physical device that aims to connect to and exchange information with other devices over the Internet. Worldwide-deployed devices are taking measurements of real-world elements and transmitting them to other entities or ecosystems to be processed and potentially analyzed. Hence, the Interoperability of Things definition can be drawn directly from joining that two definitions.

Things have the ability to communicate at least with their own ecosystems. However, we find a rather fragmented situation in the field of IoT communications and information management. Where, in general, devices have their own properties, data format, and communication technologies. This makes it difficult to share information with other systems. As said in Elkhodr et al. [2], there is a need to enable interoperability, which will allow different devices to communicate and collect information with other related smart devices or systems. This need is seen dramatically when systems are made up of a variety of heterogeneous devices and geographically distributed, as in the Computing Continuum [10].

Noura et al. [4] define how to distinguish interoperability levels. Thus, they distinguish the following levels: *device*, regarding output capacity and communication protocols; *syntactic*, regarding data format, schemas, and interfaces; *networking*, depending on the network protocols being used; *platform*, according to the operative system and programming language being used; and *semantic*, taking into account the data and information models. Our work aims to improve interoperability at the semantic level. The interoperability of devices at this level allows abstracting from the particular syntax and data formats of the different devices and providing common semantics to all managed data.

Interoperability and heterogeneity in the IoT and the Computing Continuum can be improved by means of generic solutions that apply to a wide range of possible applications. Looking at the interoperability at the platform level, oneM2M [11] specifies the IoT/M2M service layer platform as a basis for developing IoT/M2M applications. It also proposes an ontology for communicating sensing data but that does not take into account actuations nor awareness rules. OPC-UA [12] defines a standard for data exchange at the platform level, but does not specify semantics for sensing and actuation. BigIoT [13] that employs interoperability patterns to enable cross-platform interoperability but only in a very simple way. Other solutions propose complete frameworks, such as Pusztai et al. [14], which generates a boilerplate code for IoT devices from models to improve interoperability. However, the treatment of ontologies and data semantics is not fully integrated in the proposal.

2.2. Ontologies and semantics for the interoperability of things

An ontology is an abstract interpretation of the concepts, data, and characteristics being managed in a domain. As described in Noy and McGuiness [15], ontologies provide several benefits: (1) share a common understanding of the structure of information among software agents; (2) enable the reuse of domain knowledge; (3) domain assumptions are made explicit; (4) domain knowledge can be analyzed. Leveraging all these benefits is key for interoperability. As stated in Bittner et al. [16], Jasper and Uschold [17], ontologies facilitate semantic interoperability between humans, computers, and systems. They consider them as an enabler to achieving communication interoperability among software systems.

Avancha et al. [18] proposed, to our knowledge, the first ontology for wireless sensor networks with the aim of improving the interoperability of devices, using OWL-Lite to allow modification of sensor parameters based on different criteria (environmental, power, etc.). Eid et al. [19] proposed an ontology to address the problem of data heterogeneity in sensor networks.

The Open Geospatial Consortium (OGC) published a series of standards for the Interoperability of Things. The Sensor Web Enablement was designed for interoperability of sensor and actuator systems. O&M[1] defines standard XML Schema models for observations and features involved in sensor measurements. SensorML[2] provides robust and semantic means of defining characteristics and capabilities of sensors, actuators, and computational mechanisms.

The Semantic Sensor Network Incubator Group (SSN-XG) developed an ontology to describe sensors and network resources [20], to semantically enable applications interoperability, the Sensor and Sensor Network (SSN), aiming to answer the need for a domain-independent and end-to-end model for sensing applications by merging sensor-focused (SensorML), observation-focused (O&M), and system-focused views. The World Wide Web Consortium (W3C) recommended to use an updated version of the SSN Ontology [21] when the Semantic Web and Linked Data technologies were needed. It is the SSN/SOSA (Sensor, Observation, Sample, and Actuator) ontology, a lightweight but self-contained core ontology of SSN.

In addition to SSN, other ontologies empower the Interoperability of Things. The NGSI-LD ontology [22], an ETSI standard that adds context information for schema information sharing, but does not support actuation environments. SAREF [23], an ETSI-standardized ontology for IoT Smart Appliances. It includes and also empowers elements for the monitoring and control of entities. However, the ontology is not general and has a specific domain. Kiljander et al. [3] proposes the interoperability of semantic information brokers using the IoT-A[3] project. Although it supports the sensor and actuation part, it does not consider the context management that enables modeling context-awareness rules.

1 Observations and Measurements (O&M): https://www.ogc.org/standards/om.
2 Sensor Model Language (SensorML): https://www.ogc.org/standards/sensorml.
3 IoT-A: https://www.iot-a.eu.

IoT systems also require the ability to execute actions whenever certain conditions are met, which implies understanding the data and its structure alongside the environment, in order to know when to react accordingly. To enable these actuation capabilities, it is necessary to obtain context information and the primary way of achieving this is through context-awareness mechanisms. Moreover, these systems have to be able to determine when there is a data anomaly [24], to be aware of what is happening. In Xue et al. [25], a semantic sensor network is described, aiming at a context-awareness process, although we find that the proposal is not complete in terms of managing the sensor network. Alirezaie et al. [26] propose a context-aware housing system using semantics, but it applies only to a specific domain. Sehic et al. [27] develop a programming model for context-awareness-related applications for large-scale pervasive systems, but that does not take the semantics of entities into account.

2.3. Interoperability in the computing continuum

The Computing Continuum paradigm is promising the best of each computing tier. However, there are only a few proposals incorporating Interoperability of Things to this paradigm. Taherizadeh et al. [28] develop a semantic model to achieve interoperability between microservices. They claim that their model can be adapted to any context but the work is based on a three-layer architecture that limits its possibilities in the Computing Continuum. Other proposals focus only in a specific domain. For example, Hastbacka et al. [29], they propose a solution for industrial cyber–physical systems and Mahmud et al. [30], focus on the interoperability of services in health applications. More broadly, Zeng et al. [31] defined an ontology for data-intensive systems in the IoT, building a semantic model for IoT data management in the Edge–Fog–Cloud infrastructure, but they do not handle neither context-awareness nor actuation.

Although our proposal is domain independent, we review here related work in the automotive domain because it is the one we use for experimentation. Existing proposals are mostly limited to proof-of-concept applications. Then, Zhao et al. [32] develop an ontology for driving decision making, used mainly to represent knowledge maps and possible driving routes for autonomous vehicles but it does not take into account the external elements of the vehicle. Viktorovic et al. [33] proposes an ontology for intelligent urban traffic systems focusing on connected autonomous vehicles. They extend SOSA for integrating large volumes of time-sensitive data from sensor platforms and establish a basis on which the need for more control over the data layer was evident. However, the resulting ontology focuses on the data, but not on the elements used to obtain it or where it has been observed. Klotz et al. [34] sketch an ontology for vehicle signals based on SSN/SOSA. The base ontology is close to the one we empower in this work. However, they restricted its use to the automotive domain, leaving no room for other domains.

2.4. Related work summary

We conclude from our analysis that there are still open problems on interoperability in the IoT and the Computing Continuum. They are mainly related to improving interoperability between devices in the sensing and the actuation fields towards homogenization of data and interfaces. Semantics and ontologies have been proposed as the best alternative to improve this interoperability. This is why we frame our work on the development of semantics to facilitate decision-making for better data exchange and management, especially in use cases where different types of network communication can be used.

The ontology we propose in this paper is an extension of the *Connectivity Management Tool Semantics (CMTS) Ontology* by Vila et al. [35], which was aimed at monitoring physical infrastructures by means of different IoT devices in low-power wide-area networks. The *CMTS Ontology* was originally specified as an extension of the Semantic Sensor Network (SSN) [20] and the Sensor–Observation–Sample–Actuator (SOSA) [21] ontologies. We will discuss provide in Section 4 about the motivation and the advantages provided by the reuse and extension of this ontology and its benefits when applied to autonomous vehicles.

3. The IoT Monitoring and Actuation ontology (IoTMA)

We propose the IoTMA ontology (IoT Monitoring and Actuation ontology) as a step forward towards a solution for the problems of the Interoperability of Things and the Computing Continuum since it is aimed at allowing tools for monitoring entities in the world through the use of IoT devices in Computing Continuum applications. The IoTMA ontology is independent of any particular domain and, thus, all software systems able to handle the components specified in this ontology will be able to handle all different IoT deployments according to this ontology without having to modify any single line of code and independently on whether this deployment has been performed. The only thing that will have to be developed is how do the different devices provide the ontology with their information.

This section is structured as follows. First, we provide an overview of the concepts in the ontology and how they are related to each other. Then, we explain all these concepts in detail. We explain later how IoTMA contributes to the Computing Continuum. Finally, we provide the resource URIs in which we base our ontology.

3.1. The IoTMA ontology

IoTMA is proposed as an extension of some existing ontologies covering different but complementary aspects that need to be taken into account to provide a solution to the problem under consideration. Thus, the CMTS ontology [35] has been used as the main conceptual core for IoTMA. SSN/SOSA [21] has been used as a core ontology for the observation of measurements and the actuation of some elements. GeoSPARQL [36] and OWL-Time [37] have been used to describe geographical locations and temporal properties, respectively. These extensions are explained in more detail in Section 4.

The IoTMA ontology is shown in Fig. 1 by means of a UML class diagram that specifies the structure of a system by showing the system's classes, attributes, methods, and the relationships among objects. IoTMA is based on two key concepts: *Thing*, the IoT devices that support monitoring and actuation; and *FeatureOfInterest*, i.e. the properties, characteristics, or features to be considered for monitoring and actuation. Other key aspects of the ontology are *Observation* and *Actuation*, which are the main actions that the ontology allows to tackle.

We also provide in Fig. 1 namespaces to state the ontology from which the concepts in IoTMA come from. Thus, for example, *(SOSA:Platform)* states that the concept *Platform* in IoTMA is the same as the one in SOSA with the same name. *iotma* is the default namespace when no namespace is provided. All concepts in the IoTMA ontology will be explained in detail in Section 3.2.

In addition to the concepts specified in Fig. 1, the ontology also includes two textual integrity constraints that state conditions that any instance of the ontology must satisfy. These constraints are needed to accurately model domain's context-aware rules: "**C1** - *Location* must have at least one *Feature* or one *Geometry*". In addition to "**C2** - *Observation*'s *Time* must be equal to or before *Actuation*'s *Time*".

3.2. Description of the IoTMA concepts

3.2.1. Platform

A *Platform* represents any subset of the elements that are used to observe the world, elements that are monitored and eventually acted upon. We have extended the SSN/SOSA Ontology in order to provide a basis for representing this concept. In particular, the *sosa:Platform* (OWL-Class), which is defined there as "An entity that hosts other entities, particularly: Sensors, Actuators, Samplers, and other Platforms". It is in charge of hosting (*sosa:hosts*) different devices (*Thing*s) but also applying the reflexive conditions of the *sosa:Platform* in terms that a *Platform* can *host* another *Platform* for better group handling. For instance, a platform can be either a car, a building, a group of devices, or a group of cars, buildings, etc. It is related to *ThingStatus* to allow handling the status of a *Thing* for each *Platform*. A *Platform* is identified by a unique *name* and contains its location information to allow them to be placed.

3.2.2. Thing

A *Thing* is an element that reports information employing sensors and can have actuation capabilities. This entity is an extension of the *sosa:Platform* entity of the SSN/SOSA Ontology. As can be seen in Fig. 2, it has a type *(ThingType)*, can be one of *CarDevice*, *SensingDevice*, *TrafficDevice*, *NetworkDevice*, but is not limited to these, more types can be added. It is owned or placed (*sosa:hosted*), at least by one *Platform*. It can be located in a *Location*. A *Thing* is identified by a unique *name*. It can *host* multiple connected *Sensor*s to obtain measurements and *Actuator*s to perform actions. For instance, a thing can be a sensing module that is part of the *Platform*. In the automobile context, it could be a *Rain Module*. In a building, it could be the module responsible for the status of an elevator.

3.2.3. Sensor

*Sensor*s are elements capable of making raw measurements (*Observations*) of a given property *(ObservableProperty)*. This entity is an extension of the *sosa:Sensor* entity of the SSN/SOSA Ontology. *Sensor*s are part of the *Thing*s entities and, as can be seen in Fig. 3, are identified by a unique *name*. One *Sensor* is able to observe one *ObservableProperty*. At a certain moment of *Time* it can generate an *Observation* of an element being observed. There is also the possibility of locating a *Sensor* at a particular *Location*. The *Sensor* entity includes concepts of elements that are primarily intended to be used to monitor the logical infrastructure that supports the whole monitoring process. Furthermore, there are also *Sensors* needed to know the data related to the manufacturer, the product model, and the specifics on how they are working and communicating information. Examples of the latter are a gyroscope, a switch, pressure, etc., while examples of the former are the CPU usage of a computer processor, the network status of a device, or an image recognition.

3.2.4. FeatureOfInterest

*FeatureOfInterest*s are the elements, properties, or characteristics that are to be observed and manipulated. This entity is an extension of the *sosa:FeatureOfInterest* entity of the SSN/SOSA Ontology. *FeatureOfInterest* elements can be either *Physical* or *Virtual*, and are identified by a unique *name*. The former applies, as the name suggests, to elements that can be found in the physical world: places, road elements, cars, infrastructures, living entities, etc. The latter applies to virtual elements or entities that can be monitored, like the communication or the device status. It can be located in a *Location*. A *FeatureOfInterest* is made up of several *ObservableProperty*, these are the properties of the element to be observed. Furthermore, a *FeatureOfInterest* is also made up of several *ActuatableProperty*, these are the properties of the element on

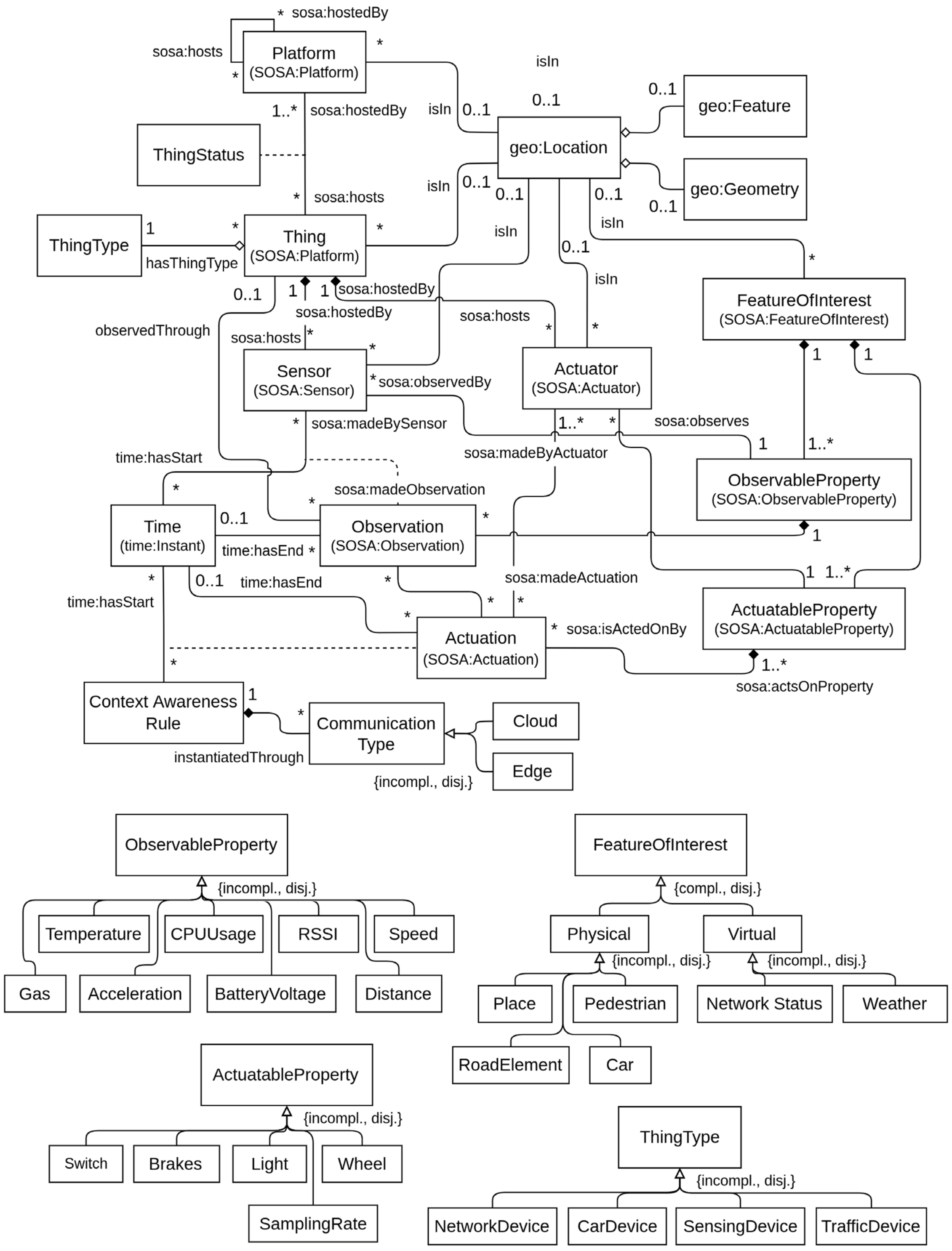


Fig. 1. Overview of the IoTMA Ontology.

which *Actuations* can be taken. For instance, this could be the status of an element, like the car status or a building's floor status.

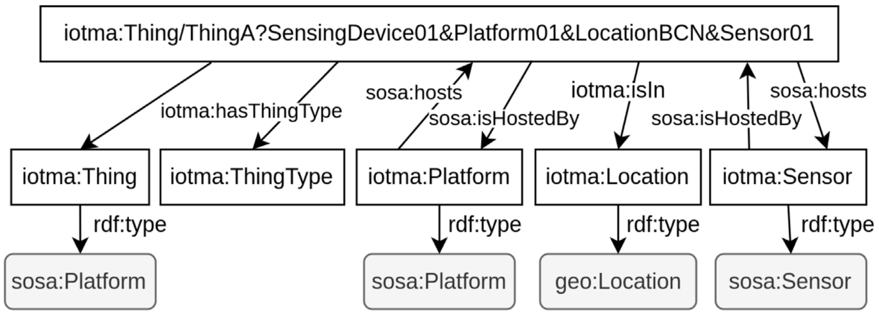


Fig. 2. Example of a Thing description.

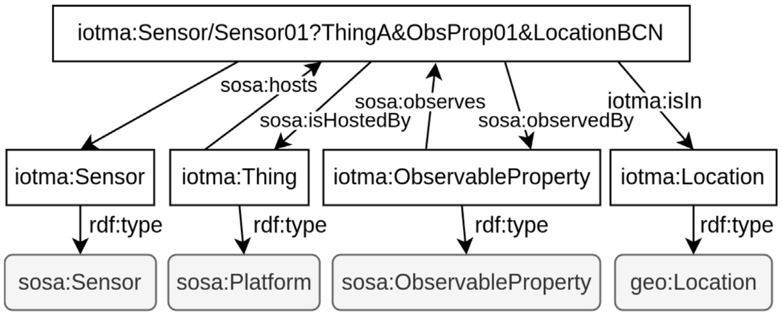


Fig. 3. Example of a Sensor description.

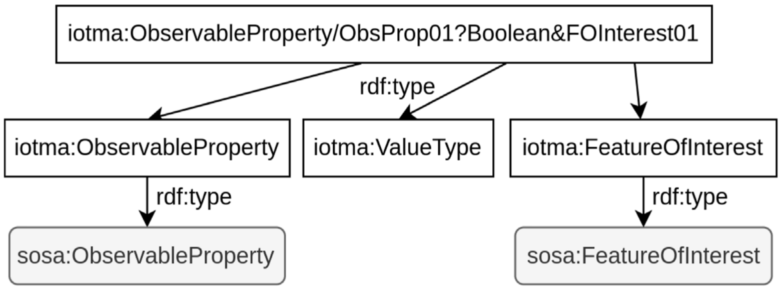


Fig. 4. Example of an ObservableProperty description.

3.2.5. ObservableProperty

ObservableProperty are the properties of the *FeatureOfInterest* to be observed. This entity is an extension of the *sosa:ObservableProperty* entity of the SSN/SOSA Ontology. These are the properties to be observed by *Sensor*s; for a given characteristic, property, or feature, and are identified by a unique *name*. We provide Fig. 4 to show its components. It has a type of technical value to read, which can be a `BOOLEAN`, `STRING`, `INTEGER`, or `FLOAT`. It is used to indicate the type of data of the *Observation* measured for a property. Examples of these properties are the remaining gas of a car, the inclination of a building, the temperature of an asset, the current speed of a vehicle, etc.

3.2.6. Observation

When a *Sensor* measures the value of an *ObservableProperty* at a given instant of *Time* (*timeStart*) it generates an *Observation*. As seen in Fig. 5, it is also possible to include a *Time* interval (*timeEnd*) to support *Observations* that cannot measure the value instantaneously. As indicated in Section 3.2.5, an *Observation* has a measured value, which must be one of the above-mentioned types. It also has *SensorName*, to whom the measurements are provided. In addition, information about the network component through which it is transmitted can be included. This entity is an extension of the *sosa:Observation* entity of the SSN/SOSA Ontology.

3.2.7. Actuator

*Actuator*s are the elements that execute *Actuations* that need to be taken from a given property *(ActuatableProperty)*, in order to change the state of an element of the world. They are part of the *Thing*s entities, identified by a unique

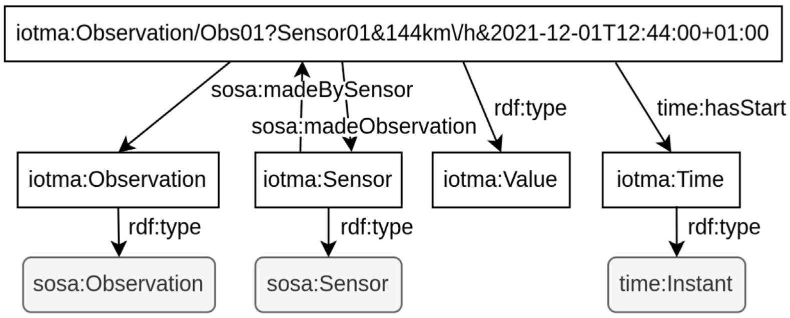


Fig. 5. Example of an Observation description.

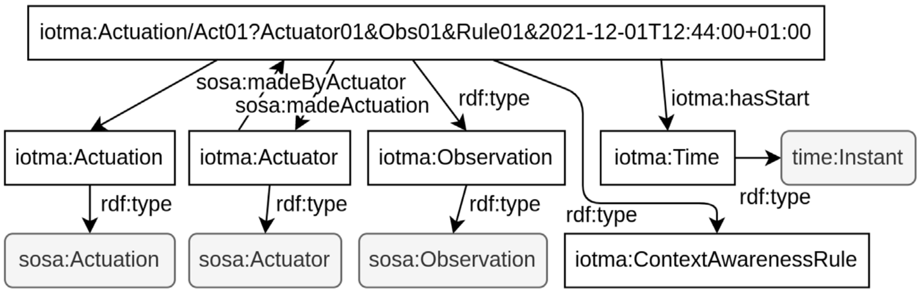


Fig. 6. Example of an Actuation description.

name. This entity is an extension of the *sosa:Actuator* entity of the SSN/SOSA Ontology. One *Actuator* is able to act on one *ActuatableProperty*. At a certain moment of *Time*, given an *Observation* and by means of a *ContextAwarenessRule*, they can generate an *Actuation* to modify the state of an element. It can be located in a *Location*. A motor, a switch, or a virtual trigger are some examples.

3.2.8. *ActuatableProperty*

ActuatableProperty are the properties of a *FeatureOfInterest* on which actions can be taken, and are identified by a unique *name*. This entity is an extension of the *sosa:ActuatableProperty* entity of the SSN/SOSA Ontology. These are the properties that will be modified when an *Actuation* takes place; for a given characteristic, property, or feature. Examples of *ActuatableProperty* are, for instance, the car's windows state, or the temperature of a car, an email to warn some agents, etc.

3.2.9. *Actuation*

ContextAwarenessRules detect corrections or actions to be taken from *Observations*, when certain conditions are met. These actions are named *Actuations*, and are carried out by the *Actuators*. This entity is an extension of the *sosa:Actuation* entity of the SSN/SOSA Ontology. *Actuations*, as seen in Fig. 6, include an instant of *Time* when it starts taking place, and if needed, an end *Time*. They also include the *ContextAwarenessRule* that triggered the particular *Actuation*, and the *observationIDs* of the observations that triggered the *Actuation*. Examples of these actuations are, for instance, changing the car's windows state, or adjusting the temperature of a car, sending an email to warn some agents, etc.

3.2.10. *ContextAwarenessRule*

ContextAwarenessRules are the rules or conditions that define how the system acts. They are responsible for reviewing the *Observations* gathered in the system for future actions, if needed. If the condition is met and is intended to trigger an action, an *Actuation* will be created and, thus, executed. They contain information about the *Sensor* they are observing, and the comparison to be performed for those *Observations*, as can be seen in the provided Fig. 7. Depending on the data type being measured, comparisons can be:

- `BOOLEAN`: Comparison `EQUALS` or `NOT_EQUALS`.
- `STRING`: Comparison `EQUALS` or `NOT_EQUALS`.
- `INTEGER`: Comparison `EQUALS`, `NOT_EQUALS`, `LESS_THAN`, or `MORE_THAN`,
- `FLOAT`: Comparison `EQUALS`, `NOT_EQUALS`, `LESS_THAN`, or `MORE_THAN`,

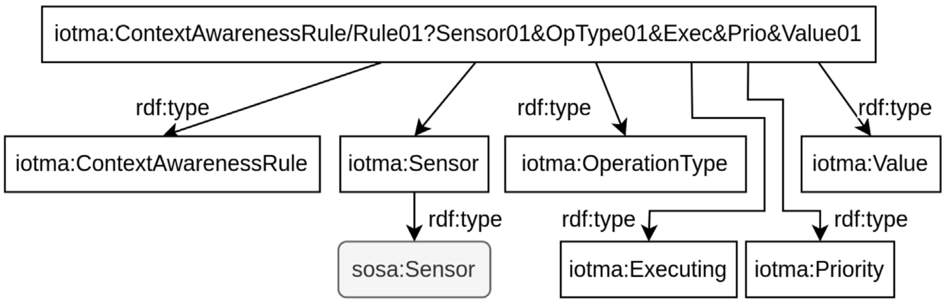


Fig. 7. Example of a ContextAwarenessRule description.

This entity is the key module for bringing decision capacity to the Computing Continuum. This module defines whether the *Actuation* to be produced has *priority* or not using a variable. *Priority* concept is a simplification to help the system decide where to send the action to execute the context rule. For instance, low *priority* actuations can be sent to be executed in the Cloud or through slow communication channels, while higher *priority* actuations will be sent to closer executors to the generated data and using the lowest latency channels available. If it has *priority* enabled, client systems will know it, enabling faster communication through Edge networks. Therefore, they will be able to decide to which system or through which network sends the information. In this way, our proposal allows assigning tasks to the Cloud or the Edge in a transparent manner to IoT management and can be executed at run-time according to the context-awareness rules defined in the ontology.

A textual example would be as follows: "If there is an *Observation* that meets the condition that a *Sensor* X reports a data from an *ObservableProperty* greater than Y, then it executes an *Actuation* Z".

3.2.11. Location

If desired, the user has the ability to indicate the location of elements in the physical world, employing a *Location*. With this purpose, the system is provided with the ability to locate *Thing*s, *Sensor*s, *Actuator*s, *FeatureOfInterest*s, and *Platforms*. For this entity, we have extended GeoSPARQL to provide a basis for its representation. Extending *geo:Location* entity and also giving the ability of defining well-known names for places (*geo:Features*) or geometry elements (*geo:Geometry*). To exemplify, a valid location can be "Vienna" as a *Feature*, but also "41.389289, 2.113168" as a *Geometry*.

3.3. IoTMA in the computing continuum

The *IoTMA* ontology describes the concepts of general monitoring and actuation terms needed to understand situations and contexts. Enabling users to develop their own usecase in the IoT. By using this proposed approach, a monitoring information system can be accessed seamlessly. We give users mechanisms to develop conceptual models for use cases in the broad Computing Continuum paradigm. This ontology allows the distribution of the execution of the applications among the different tiers of the paradigm. Furthermore, by means of context-awareness rules, it goes one step further and allows users to define actions to be performed from predefined values. This allows to distribute and decentralize the results in a clear way to the user's preference.

3.4. Resource URIs

In Fig. 1, we use URIs (Uniform Resource Identifier), a unique sequence of characters that identifies a logical or physical resource, to increase the simplicity and manageability of the systems defined according to our ontology. We show in Table 1 the main design of these URIs. The first column represents the class name of the entity in our ontology. The second represents the generated URI content. The format of the URIs follows the pattern defined below. The `BASE_URI` refers to the URL entry point, located in a test domain. The `CLASS_NAME` indicates the entity name, following the `CLASS_ID`, which is the identifier of each entity instantiation. In addition to the `CLASS_PROPERTY_N`, which contains the *N* properties of the instantiated entity.

```
{BASE_URI}:{CLASS_NAME}/{CLASS_ID}?{CLASS_PROPERTY_1}&{CLASS_PROPERTY_N}
```

4. Ontologies extended by IoTMA

As we have seen in the previous section, the IoTMA ontology is proposed as an extension of several existing ontologies which cover different but complementary aspects that need to be taken into account to provide a solution to the problem under consideration. In particular, we have used four of the existing ontologies for the following purposes:

- CMTS: ontology used as a basis for conceptual modeling.
- SSN/SOSA: ontology used to gather measurements from sensors and act using actuators.

Table 1
Summary of the main URIs in IoTMA.

Class	URI Patterns
Platform	`iotma:Platform/{PlatformName}?{LocationName}&{Platforms}&{Things}`
Thing	`iotma:Thing/{ThingName}?{ThingTypeName}&{LocationName} &{Sensors}&{Actuators}&{Platforms}`
Sensor	`iotma:Sensor/{SensorName}?{ThingName}&{ObservablePropertyName} &{LocationName}&{Observations}`
Actuator	`iotma:Actuator/{ActuatorName}?{ThingName} &{ActuatablePropertyName}&{LocationName}&{Actuations}`
FeatureofInterest	`iotma:FeatureOfInterest/{FeatureOfInterestName}? &{ObservablePropertyNames}&{ActuatablePropertyNames}&{LocationName}`
ObservableProperty	`iotma:ObservableProperty/{ObservablePropertyName}? &{ValueType}&{FeatureOfInterestName}&{Sensors}`
Observation	`iotma:Observation/{ObservationID}?{SensorName}&{Value}&{TimeStart}`
ActuatableProperty	`iotma:ActuatableProperty/{ActuatablePropertyName}? &{FeatureOfInterestName}&{Actuators}`
Actuation	`iotma:Actuation/{ActuationID}?{ActuatorNames} &{ObservationIDs}&{ContextAwarenessRuleName}&{TimeStart}`
ContextAwarenessRule	`iotma:ContextAwarenessRule/{ContextAwarenessRuleName}?{SensorName} &{OperationType}&{Executing}&{Priority}&{Value}`
Location	`iotma:Location/{LocationName}?{Feature| Geometry}`
Time	`iotma:Time/{Date}T{Time}+{TimeZone}`

- GeoSPARQL: ontology used to describe geographical locations.
- OWL-Time: ontology used to describe temporal properties.

In this section, we explain with more detail how *IoTMA* makes use and extends these ontologies.

4.1. CMTS - Connectivity Management Tool Semantics

IoTMA extends functionalities of the *CMTS* ontology, a previous work of Vila et al. [35] that aimed to monitor physical infrastructures in IoT under LPWAN, mainly LoRa and NB-IoT. *CMTS* was originally specified as an extension of the World Wide Web (W3C) and the Open Geospatial Consortium (OGC), Semantic Sensor Network ontology (SSN) and Sensor–Observation–Sample–Actuator (SOSA) ontology.

CMTS incorporates terms such as *Site* for grouping *Devices*. *Devices* can be one of *Gateway*, *Node*, or *Sensor*. *Gateway*s aim to report data packets to the *Sites*. *Node*s act as a box for grouping *Sensors*. *Sensor*s are in charge of the *ObservationProperty* to be measured, resulting in *Observation*s.

The main differences between *IoTMA* and *CMTS* are explained below.

4.1.1. Adding actuation capabilities

CMTS was only intended to monitor values obtained from *Sensors*. Now, *IoTMA* also provides the capability to act on certain predefined properties, if necessary. This capability is an important extension, as required by the problem addressed in this article. The following changes have been applied for this purpose:

- *Thing*s now are composed of *Actuators*. Thus, allowing to have actuation/intervention elements.
- *ContextAwarenessRule* entity is introduced. It has the goal of defining rules that the system must check to be able to act in case the established conditions are met. These rules are triggered by a temporal element or through an *Observation*.
- *ActuatableProperty* entity is proposed, with the objective of grouping properties that can be acted upon.

4.1.2. Device to thing

A *Device* was considered an abstract concept that could be completed as a *Gateway*, a *Node*, or a *Sensor*. To make the ontology more general, the following changes have been made:

- *Gateway* concept disappears in favor of *Thing*: With the aim of supporting more use cases outside of the *LPWAN* network topic. This implies that a *Thing* can be monitored in sensing terms but also generate actuations. In *CMTS*, a *Gateway* could only have *SoftwareSensors*, which were to analyze KPIs of the *Gateway* itself. Now, a *Gateway* is a *Thing*. Therefore now it can host any type of *Sensor*.

- *Node* concept disappears in favor of handling the information in the *Thing* entity itself. The *Sensors* that a *Node* could have are now absorbed by the *Sensor*s connected to a *Thing*.
- *Sensor* entity continues having the same naming, but is no longer considered a *Thing* anymore. It is a part of a *Thing*. With the changes applied at the *Thing* level, we have considered and proposed the simplification of the *Sensors* typology. Therefore, leaving only one category level and removing the *HardwareElement* and *SoftwareElement* concepts.

4.1.3. Incorporating FeatureofInterest

Just as there are parts that have been simplified, the knowledge provided has been extended in this case. From the SSN/SOSA definitions of the groups of *ObservableProperties* and *ActuatableProperties*, we have introduced the concept *FeatureOfInterest*. In *CMTS*, a single level of properties to be observed was initially sufficient. But after testing and adding the elements to act on, we have decided to include this grouping concept as well, to enable better handling of the monitored elements in general.

4.1.4. Site to platform

Both elements continue to have the same conceptual purpose. However, the term now used is *Platform* since it is a more general name and easier to understand. In addition, the conceptual model has been provided with the possibility for a *Platform* to host to another *Platform*, thus allowing multi-level groups.

4.2. Semantic Sensor Network and Sensor–Observation–Sample–Actuator - SSN/SOSA

SSN/SOSA [21] is still the ontology that comes closer to the concepts here being modeled. This is why it continues to be used as a basis of the project. It provides a lightweight core for defining common classes and properties of data being managed in the IoT domain. It supports both sensing and actuation capabilities, which still provides a perfect fit for modeling interoperability in the IoT and in the Computing Continuum. The SSN/SOSA classes that have been (re)used in our solution are:

- *sosa:Platform*, to describe:
  - *Platform*, representing subsets of elements that are used to observe the state of the world.
  - *Thing*, an element that reports information and can also have actuation capabilities.
- *sosa:FeatureOfInterest*, to specify the superior element of a particular *ObservableProperty* or *ActuatableProperty*.
- *sosa:Sensor*, to describe the low-level *Sensor*s that are collecting information.
- *sosa:ObservableProperty*, to specify the elements to be observed by a *Sensor*.
- *sosa:Observation*, to provide measurement values of an *ObservableProperty* taken by *Sensors*.
- *sosa:Actuator*, to describe the low-level *Actuator*s that are in charge of executing *Actuations*.
- *sosa:ActuatableProperty*, to specify the elements that can be modified by an *Actuator*.
- *sosa:Actuation*, to provide modifications of elements *ActuatableProperty* executed by the *Actuators*.

The *SSN/SOSA* is also used to define some of the properties to give more clarity to the actions performed: *sosa:hostedBy*, *sosa:hosts*, *sosa:observedBy*, *sosa:observes*, *sosa:madeBySensor*, *sosa:madeObservation*, *sosa:madeByActuator*, *sosa:madeActuation*, *sosa:isActedOnBy*, and *sosa:actsOnProperty*.

4.3. GeoSPARQL

We keep having GeoSPARQL [36] ontology to define the spatial elements, since it is the one that better fits our approach. It allows defining elements employing common text (*geo:Feature*) through the Well-Known Text representation of geometry (WKT), for instance, "Barcelona". In addition to using geometric points in Geographic Markup Language (GML), such as points, lines, or polygons. Using the element *geo:Location*, we define the location of *Platform*s, *Thing*s, *Sensor*s, *Actuator*s, and *FeatureOfInterest*s.

4.4. OWL-Time

As in *CMTS*, OWL-Time [37] is used here to describe temporal concepts and properties. *Time* is still used to define when there is an *Observation* (*time:hasBeginning*[4]). It is also now used to define when there has been an *Actuation*. In both cases, it is also possible to indicate an end time (*time:hasEnd*), if necessary. In addition, by means of *time:Instant* there is the possibility of defining the precise instant of occurrence.

[4] In *IoTMA*, *time:hasBeginning* renamed to *time:hasStart* for clarity purposes.

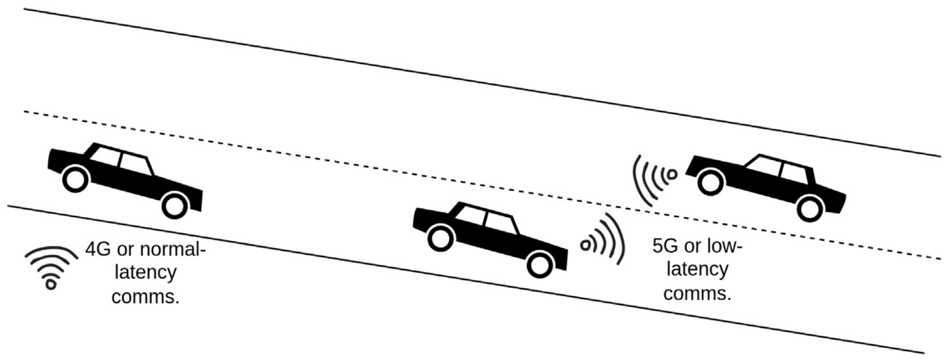


Fig. 8. Default state - normal-latency networks (left)/High-priority state - low-latency networks (right).

5. Experimentation

We have made some experiments to show the feasibility of using an ontology as means of improving interoperability information in the IoT. Our approach takes the advantages that ontologies provide when applied to the Computing Continuum paradigm. With this purpose, we have set the basis for our experimentation in the Interoperability of IoT devices in autonomous cars, under the Computing Continuum scenario. Using the entities defined in our ontology, we handle data interoperability for sensing and actuation devices. We would like to remark that the main goal behind our experimentation is that of showing that using an ontology that includes the multi-tier concept enables the Computing Continuum from that level, easing in practice all following aspects of the design and development of the system, and not that of performing an efficiency analysis regarding whether it is faster transmitting data directly or through the Cloud, which is clearly out of scope of this work.

Our practical use case is illustrated in Fig. 8. We aim to communicate information through different types of network, depending on the value of the data being collected and analyzed. These data are collected by vehicles to detect obstacles and thus prevent accidents. With this, we enable the monitoring of dangerous elements that can be present on the roads.

The figure shows that the vehicle on the left does not detect obstacles, and then its communication is by default a standard network. On the right, there are two vehicles that are about to cross each other. Therefore, they should need to communicate with each other through a faster network for low-latency critical communications. Both the communication among vehicles and the choice of the network to use are performed automatically because of the knowledge endowed in the IoTMA ontology.

Our use case also takes into account the different computational tiers of the Computing Continuum, selecting the network to which submit the data (i.e., Cloud or Edge), depending on the context of the measurements made. The experimentation code accepts well-formed information using the semantics established by our proposed IoTMA ontology. Thus, entities communicating in this experimentation are using the conceptual model of our ontology and applying it to the generated messages. Measurements are gathered, analyzed, and presented up to the point of generating an environment for third-party systems to read the resulting information through an HTTP API.[5] The ecosystem is able to internally manage the monitoring and the possible reactions needed, depending on the priority of communicating this information through different mobile networks.

5.1. The setup

We made our experiments using two Raspberry Pi 4B.[6] as a simile of two vehicles. This is shown in Fig. 9 The left part of the figure shows the real setup of the experiment. The right-hand part shows the conceptual overview of this setup. One Raspberry Pi has a critical sensor (exemplified as a light sensor) and an LED to indicate if some action is communicated. The other one has an actuator that consists of a sound buzzer and a LED to visually know when an action is executed. On top of both Raspberry Pi devices, there is a GrovePi shield,[7] which is used to wire the sensors to each device. Both devices are able to reach the Internet via a WiFi router (black cube in Fig. 9a), to enable Raspberry Pi devices to update their entities with the available data on the Cloud. Both devices are also wired via Ethernet cables to a router (black flat device in Fig. 9a). Thus, we create a local area network to simulate an Edge communication which is isolated from the Internet.

5 API: Application Programming Interface.

6 Raspberry Pi: Small computers with limited computing capabilities. https://www.raspberrypi.org.

7 GrovePi Shield for Raspberry Pi: https://www.seeedstudio.com/GrovePi.html.

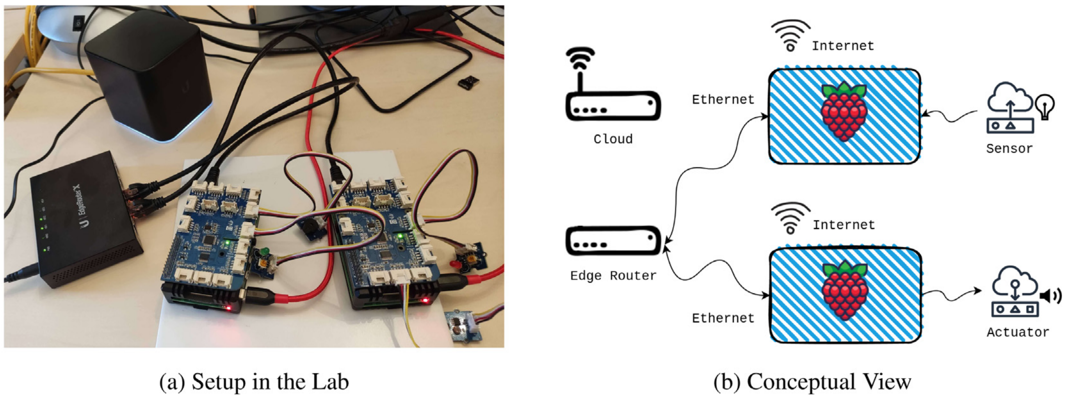


Fig. 9. Experimentation setup.

Our Cloud setup consists of a *GCP E2-small* instance that has 1 vCore and 2 GB of RAM under Debian 11 Linux. In this instance, we have deployed our own server code, developed by us, in accordance with the proposed IoTMA ontology. In Section 5.4, a more in-depth explanation of our code is provided. With this *GCP E2-small* instance, the server load is between 5% and 20% of CPU when accessing or updating information from Raspberry Pi devices, and 500 MB of RAM are used.

5.2. What does the experiment do?

One Raspberry Pi is taking measures (*Observations*) of ambient brightness,[8] and sending them to the Cloud. Every few seconds, the Raspberry Pi updates its *ContextAwarenessRules* for later use. As the Raspberry Pi is a *Thing* that has a *Sensor* it knows its property to observe (*ObservableProperty*) and to report (*Observation*). The Raspberry Pi also knows its context-aware rules and it is able to execute them locally. If a rule is triggered at some point, the Raspberry Pi will create an *Actuation*. Moreover, and depending on the criticality of the information being monitored, it has to decide whether it is communicated directly via the local-network router to the other Raspberry Pi (emulating the other vehicle) or it can be sent to the Cloud for a later transmission to other *Things* (i.e., vehicles). In our setup, this decision is made by taking into account the amount of light received in one Raspberry. As we mentioned, we use a simile that would be abstracting the vehicles proximity with the amount of light received, in this sense, no light sensed is having a car in the proximity. As a result, if they are close, the actuation to be performed is treated as a priority, which means sending the actuation through an Edge connection, via Ethernet, to the Raspberry Pi devices using HTTP protocol,[9] or via WiFi to the cloud, using WebSockets.[10]

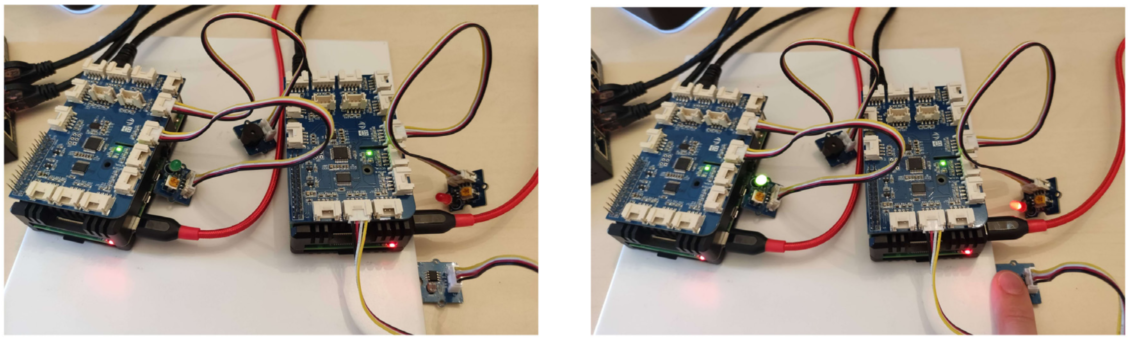
In Fig. 10a we show the idle state of the system where the light sensor is reading values of the amount of light it receives. Every few seconds, the device is updating, via the Cloud, the context-awareness rules that might be triggered through its sensors. Previously, the *Thing*, the *Sensor*, and the *ObservableProperties* that correspond to the use case have been registered in the Cloud. At the same time, the *Sensor* obtains a measurement (*Observation*) and sends it to the Cloud.

When the *Thing* detects that the *Sensor* has an active *ContextAwarenessRule* that complies with the established criteria, and is enabled, it generates an *Actuation*. This *Actuation* will be sent via the Cloud or via Edge, depending on the *ContextAwarenessRule* configuration. In Fig. 10 Raspberry Pi devices are close to each other and an actuation has been transmitted through the Edge router, generating peer-to-peer communication. Fig. 10b shows how the left Raspberry has an active green LED corresponding to the activated alarm.

The difference between the traditional (or Cloud-centered) way of communication, which also applies to IoT, and the Edge one, is shown in Fig. 11. In the traditional way (top of the figure), the IoT device takes a measurement and sends it through a communication point to the Internet (Router A in this example). This router sends the information to a Cloud server, which, by making *N* router hops, generates the resulting actions. These actions are returned to the use case to another Router (which can be the same as when sending, i.e. Router A or a different one), and this one is in charge of communicating to the other device the action to be taken.

However, thanks to the Computing Continuum paradigm, both options are available. Hence, the device can take the Edge-based communication schema. In this sense, the IoT device takes a measurement and sends it to a router that acts as an Edge device. This router (or another router on the local network) transmits the information to the other

[8] We use the ambient brightness as an approximation of comparing the physical proximity of two vehicles.
[9] HTTP: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.
[10] WebSocket: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.



(a) Experiment in default state (b) Experiment when *Actuation* is triggered

Fig. 10. Experimentation results.

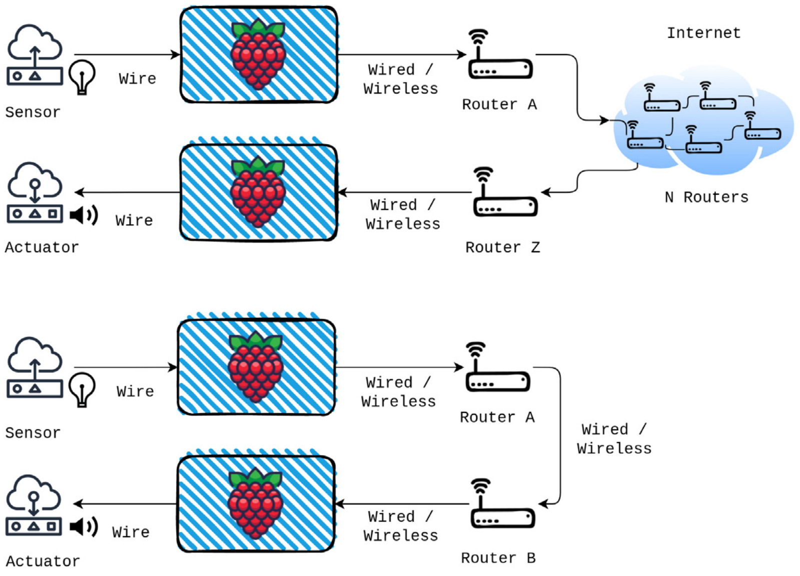


Fig. 11. Traditional way of communication (top)/Edge-based communication (bottom).

device, including the action to be taken. Consequently, it can act faster than the traditional way, since the information is not sent to the Internet. Instead, it is sent through local networks, which means *thing–thing* communication and not *thing–cloud–thing*, as traditionally.

Our solution is also more reusable since any IoT ecosystem defined in terms of our ontology can be directly handled by our solution, which does not depend on the specific technological ecosystem (as it happens with current proposals) but on the semantics of the different devices endowed in the ecosystem. In addition, it also provides the semantics to communicate in any of the aforementioned approaches.

5.3. *Results of our experiments*

From our experiment, the first objective validated is that the ontology design enables developers to communicate information between different IoT devices and systems. In addition, we have performed several tests to validate that our ontology permits modifying the destination communication endpoint as theoretically stated. This validation has been done by obtaining time metrics and comparing both forms of communication supported. For this purpose, we have set up our Cloud with the software elements to communicate with our ontology. Devices communicate via HTTP (POST + GET requests) to our Cloud backend, using JSON as in Listings 1, 2, 3, and 4. In our experimentation we proposed an automotive-related use case, although for simplification purposes, we made an analogy using a light sensor a simile of the proximity sensor, as we have already indicated in Section 5.2.

In the following, we list the steps mentioned for the evaluation and the entities that are created:

1. A *ThingType* named "Car".
2. A *Thing* named "CarOne", visible in Listing 1, with "Car" as *ThingType*.
3. A *FeatureOfInterest* named "MathematicalOperation".
4. An *ObservableProperty* named "DistanceBetween", visible in Listing 1, with "MathematicalOperation" as *FeatureOfInterest*.
5. A *Location* named "Barcelona-Sants", visible in Listing 1, pointing to the 41.38102, 2.14177 (lat,lng) coordinates and takes into account the *GeoSPARQL* ontology, complying with the *geo:Point* class.
6. A *Sensor* named "CarProximitySensor", visible in Listing 1, with "CarOne" as *Thing*, "DistanceBetween" as *ObservableProperty* and "Barcelona-Sants" as *Location*.

At this point, the *CarProximitySensor* is correctly configured to receive measurements. When this happens, for each gathered measurement, it will create an *Observation*, as stated in Listing 2, that includes the reading in the *value* field and its *measurement time*, the latter, as stated in the ontology, is an extension from *time:inXSDDateTimeStamp* (class *Time:Instant*) from the OWL-Time ontology.

Listing 1: JSON body for creating a Sensor

```
{
  "name": "CarProximitySensor",
  "thing_name": "CarOne",
  "observable_property_name":
    "DistanceBetween",
  "location_name": "Barcelona-Sants"
}
```

Listing 2: JSON body for creating an Observation

```
{
  "sensor_name": "CarProximitySensor",
  "time_start":
    "2022-04-19T14:58:04+00:00",
  "value": 60
}
```

Moreover, to give the system the ability of reacting to events, we have setup two *ContextAwarenessRules*: Listing 3 shows the configuration required to create a *ContextAwarenessRule* to notify when a high priority condition is met in the received *Observations*. In addition, a Listing 4 has also been set up showing the configuration to notify when normal priority data is observed. Then, the involved client, where the measurements are being taken, is able to send *Observations* to the Cloud, as indicated in the Listing 2, containing a well-formed message complying with the ontology. In this case, this *Observation* represents the proximity to another vehicle. Although, if the *Observation* complies with the *ContextAwarenessRule* described in Listing 3, then communication will be carried out over high-priority networks, as stated in the corresponding *ContextAwarenessRule*.

Listing 3: JSON body for creating a Context Awareness Rule with high priority

```
{
  "name": "Rule -
   When cars are close",
  "sensor_observed_name":
    "CarProximitySensor",
  "operation_type": "LESS_THAN",
  "value_to_compare_integer": 20,
  "executing": true,
  "priority": true
}
```

Listing 4: JSON body for creating a Context Awareness Rule with normal priority

```
{
  "name": "Rule -
   When cars are not close",
  "sensor_observed_name":
    "CarProximitySensor",
  "operation_type": "MORE_THAN",
  "value_to_compare_integer": 20,
  "executing": true,
  "priority": false
}
```

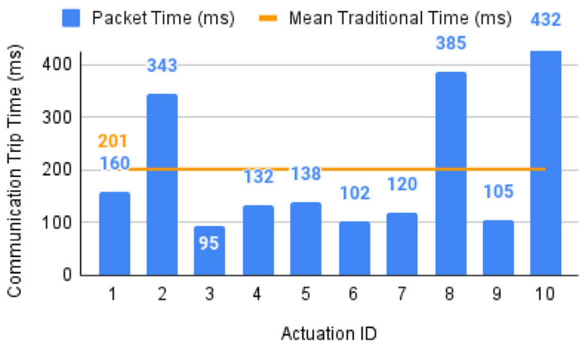


Fig. 12. Traditional-based communication time.

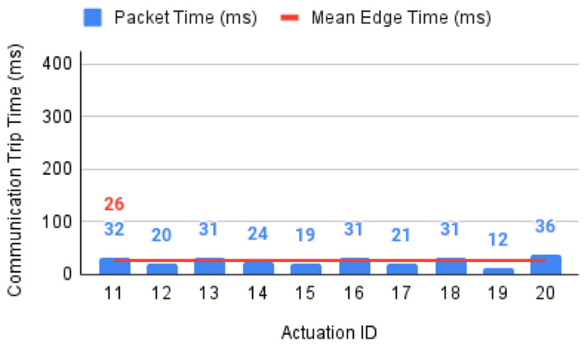


Fig. 13. Edge-based communication time.

In Figs. 12 and 13, we can observe a series of actuations that have been transmitted from one Raspberry Pi to the other. In each graph we can see, at the Y-Axis, the *Communication Trip Time (CTT)* in milliseconds. *CTT* is defined as the time difference ($\Delta t_{ctt} = t_{end} - t_{initial}$) between the instant in which the communication packet is generated ($t_{initial}$), and the instant when the packet is received by the other Raspberry Pi (t_{end}). The X-axis represents the identifier of the transmitted actuation.

Fig. 12 shows the transmission time when the environment does not detect any urgency. We can observe transmissions between 95 and 432 ms, making a mean of 201 ms, and a median of 135 ms. The observed deviation is due to the instability of the standard WiFi technology used in our tests.

In Fig. 13 the system has detected some urgency so that the communication must be done as quickly as possible. In this case, the transmissions are made via Ethernet cable and they last between 12 ms and 36 ms, making a mean of 26 ms, and a median of 27 ms, with almost no deviation.

In both figures, the flat horizontal line represents the mean time for that type of communication, which is 201 ms in the first case and 26 ms in the second.

5.4. The code

The experimentation Cloud code is built upon, and extended, from our previous work in Vila et al. [38], where we proposed a prototype to monitor IoT devices to automatically react through alarms located in the Cloud when data exceeds a certain threshold. In this paper, we have provided the system with semantic properties for data interoperability, extending from *CMTS* and SSN/SOSA, according to what we have already explained.

The client code, executed on the Raspberry Pi devices, has been built from scratch for this work and it is able to obtain real measurements from sensors, communicating with the Cloud server to obtain the ontology information, and sending the observations taken to the Cloud. Moreover, it synchronizes context-awareness rules to execute them, if necessary. In the actuation part of the code, it takes into account the measurement read and the actions to be performed. This results in an *Actuation* if the measurements comply with a context-awareness rule, thus communicating with other devices or systems that can be located in the different network tiers, Cloud or Edge. As previously mentioned, part of the code has been reused and is still in the Cloud. However, further work has been done to add the improvements mentioned in the previous sections, especially in the parts that support the detection of anomalies and the actions to be performed.

The project code is open source and is available on GitHub at https://github.com/worldsensing/iotma-edge-to-cloud-sensing-actuation. The code, located in folder `cloud_api`, is based on a microservice architecture and can be executed with minimal changes using Docker. It has been developed using Python Flask and PostgreSQL. A Swagger is also provided, to have a better understanding of the data being transmitted between services and the Cloud code and python client. The Cloud code has been tested on Ubuntu 20 and Debian 11. The `python_client` has been also tested under Ubuntu 20, Debian 11, and Raspberry Pi OS 5.1 for Raspberry Pi 3 and 4 devices.

6. Conclusions and future work

We have proposed IoTMA, an ontology for the sensing and actuation IoT domain. It describes the concepts of general monitoring and actuation terms needed for understanding situations and contexts. This includes both the conceptual part and practical experimentation in terms of having tested the correct functioning of the system. Enabling users to develop their own usecase in the IoT.

The proposed ontology consists of two main parts. On the one hand, the definition of the concept of monitoring (*Thing*) and elements to monitor (*Sensors*), while providing them with the possibility of granularizing to some extent parts of elements and employing their properties (*Observable/Actuatable Property*). On the other hand, it also includes the *Actuator*s as a *Thing*, improving previous work and developing context-awareness capabilities, allowing users to define actions to be performed from predefined values.

We give mechanisms to develop conceptual models for use cases in the broad Computing Continuum paradigm. Our proposal allows for the distribution of the application among the different tiers of this paradigm. Using context-awareness mechanisms, its execution can shift towards any place on the Computing Continuum, depending on the application needs and given by the observations of the environment.

A use case emulating autonomous vehicles has been used to remark that IoT in the Computing Continuum still needs to empower data interoperability and an ontology to make the best of the promised applications. This use case has been exemplified with an experiment to demonstrate the functionalities supported by the ontology proposed, from data collection to the execution of a high-priority actuation.

Finally, a ready-to-use system has also been provided with automated deployment for the Cloud part, including the backend with an orchestration under a microservice architecture. Additionally, a client code that handles the remaining part of the functionalities has been provided. Consisting in a ready-to-use system that works by only changing the desired IP addresses to communicate with.

Future work involves incorporating a more sophisticated notification system at the semantic level, in which users could define where the information is being sent, choosing from API REST calls, or queued protocols (such as MQTT, CoAP), emails, or even phone messaging. Furthermore, the context-awareness system could be extended to include other entities such as *Things*, *Platforms*, or other entities that could complete the possibility of monitoring each-and-every entity in the system. At a technical level, giving more possibilities to improve the precision of the queries to be executed by the *ContextAwarenessRules*, as well as to the typology of data in the *ObservableProperty*. Defining custom data classes to analyze and monitor, in addition to custom formulas to apply to the data being ingested. We also plan to incorporate the concepts explained here in the operational part of Worldsensing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We would like to thank Maria-Ribera Sancho for her useful comments on previous versions of this article. Thanks also to P. Frangoudis and A. Knoll for their help setting up the experiments. This work is partially funded by: Industrial Doctorates (2019 DI 001) from Generalitat de Catalunya. The SUDOQU project (PID2021-127181OB-I00) from MCIN/AEI. FEDER "Una manera de hacer Europa"; and project 2017-SGR-1749 from Generalitat de Catalunya. Also with the support of inLab FIB at UPC and Worldsensing. We also thank the anonymous reviewers for their valuable comments.

References

[1] Statista, Number of internet of things (IoT) connected devices worldwide from 2019 to 2030, 2020, URL https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ [Last accessed 03 Nov 2021].
[2] M. Elkhodr, S. Shahrestani, H. Cheung, The internet of things: New interoperability, management and security challenges, Int. J. Netw. Secur. Appl. 8 (2) (2016) 85–102.
[3] J. Kiljander, A. D'elia, et al., Semantic interoperability architecture for pervasive computing and internet of things, IEEE Access 2 (2014) 856–873.
[4] M. Noura, M. Atiquzzaman, et al., Interoperability in internet of things: Taxonomies and open challenges, Mob. Netw. Appl. 24 (2019) 796–809.
[5] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81.
[6] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39.
[7] F. Bonomi, R. Milito, et al., Fog computing and its role in the internet of things, in: First Workshop on Mobile Cloud Computing (MCC), 2012.
[8] P. Beckman, J. Dongarra, et al., Harnessing the computing continuum for programming our world, in: Fog Computing: Theory and Practice, 2020, pp. 215–230.
[9] L.D. Xu, W. He, S. Li, Internet of things in industries: A survey, IEEE Trans. Ind. Inf. 10 (4) (2014) 2233–2243.
[10] L. Bittencourt, R. Immich, et al., The internet of things, fog and cloud continuum: Integration and challenges, Internet Things 3–4 (2018) 134–155.
[11] J. Swetina, G. Lu, et al., Toward a standardized common M2M service layer platform: Introduction to oneM2M, IEEE Wirel. Commun. 21 (2014) 20–26.
[12] W. Mahnke, S.-H. Leitner, D. Matthias, OPC Unified Architecture, Springer, 2009.
[13] A. Bröring, S. Schmid, et al., Enabling IoT ecosystems through platform interoperability, IEEE Softw. 34 (2017) 54–61.
[14] T.W. Pusztai, C. Tsigkanos, S. Dustdar, Engineering heterogeneous internet of things applications: From models to code, in: IEEE International Conference on Collaboration and Internet Computing (CIC), 2019, pp. 222–231.
[15] N.F. Noy, D.L. McGuiness, Ontology Development 101: A Guide to Creating Your First Ontology, Technical Report, Knowledge Systems - Stanford University, 2001.
[16] T. Bittner, M. Donnelly, S. Winter, Ontology and semantic interoperability, 2006.
[17] R. Jasper, M. Uschold, A framework for understanding and classifying ontology applications, in: Proceedings of the IJCAI99 Workshop on Ontologies and Problem-Solving Methods, 1999.
[18] S. Avancha, C. Patel, et al., Ontology-driven adaptive sensor networks, in: International Conference on Mobile and Ubiquitous Systems: Networking and Services (MOBIQUITOUS), 2004, pp. 194–202.
[19] M. Eid, R. Liscano, et al., A universal ontology for sensor networks data, in: IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIVEMSA), 2007, pp. 59–62.
[20] A. Haller, K. Janowicz, et al., The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation, Seman. Web J. (2018).
[21] K. Janowicz, A. Haller, et al., SOSA: A lightweight ontology for sensors, observations, samples, and actuators, J. Web Semant. 56 (2019) 1–10.
[22] G. Privat, Guidelines for Modelling with NGSI-LD, Technical Report, ETSI, 2021.
[23] L. Daniele, R. Garcia-Castro, et al., SAREF: the smart applications reference ontology, 2020, URL https://saref.etsi.org/core/v3.1.1/ [Last accessed 10 Mar 2022].
[24] C. Perera, A. Zaslavsky, et al., Context aware computing for the internet of things: A survey, IEEE Commun. Surv. Tutor. 16 (1) (2014) 414–454.
[25] L. Xue, Y. Liu, et al., An ontology based scheme for sensor description in context awareness system, in: 2015 IEEE International Conference on Information and Automation, 2015, pp. 817–820.
[26] M. Alirezaie, J. Renoux, et al., An ontology-based context-aware system for smart homes: E-care@home, Sensors 17 (7) (2017).
[27] S. Sehic, F. Li, S. Nastic, S. Dustdar, A programming model for context-aware applications in large-scale pervasive systems, in: IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2012, pp. 142–149.
[28] S. Taherizadeh, D. Apostolou, et al., A semantic model for interchangeable microservices in cloud continuum computing, in: Information 2021, Vol. 12, Multidisciplinary Digital Publishing Institute, 2021, p. 40.
[29] D. Hastbacka, J. Halme, et al., Dynamic edge and cloud service integration for industrial IoT and production monitoring applications of industrial cyber-physical systems, IEEE Trans. Ind. Inf. 18 (2022) 498–508.
[30] R. Mahmud, F.L. Koch, R. Buyya, Cloud-fog interoperability in IoT-enabled healthcare solutions, in: ACM International Conference Proceeding Series, Association for Computing Machinery, 2018, p. 10.
[31] W. Zeng, S. Zhang, et al., Semantic IoT data description and discovery in the IoT-edge-fog-cloud infrastructure, in: IEEE International Conference on Service-Oriented System Engineering (SOSE), 2019, pp. 106–10609.
[32] L. Zhao, R. Ichise, et al., Ontology-based driving decision making: A feasibility study at uncontrolled intersections, Trans. Inf. Syst. E100.D (2017) 1425–1439.
[33] M. Viktorovic, D. Yang, B. Vries, Connected traffic data ontology (CTDO) for intelligent urban traffic systems focused on connected (semi) autonomous vehicles, Sensors 20 (2020) 2961.
[34] B. Klotz, S.K. Datta, et al., A car as a semantic web thing: Motivation and demonstration, in: Global Internet of Things Summit (GIoTS), 2018, pp. 1–6.
[35] M. Vila, M.-R. Sancho, E. Teniente, X. Vilajosana, Semantics for connectivity management in IoT sensing, in: International Conference on Conceptual Modeling (ER), 2021, pp. 297–311.
[36] OGC, GeoSPARQL - A geographic query language for RDF data, 2012, URL https://www.ogc.org/standards/geosparql [Last accessed 05 Dec 2021].
[37] W3C - OWL-Time, Time ontology in OWL, 2020, URL https://www.w3.org/TR/owl-time/ [Last accessed 05 Dec 2021].
[38] M. Vila, M.-R. Sancho, E. Teniente, XYZ monitor: IoT monitoring of infrastructures using microservices, in: International Conference on Service-Oriented Computing (ICSOC) STRAPS Workshop, 2021, pp. 472–484.