A semantic-based discovery service for the Internet of Things

Porfírio Gomes, Everton Cavalcante, Thais Batista, Chantal Taconet, Denis Conan, Sophie Chabridon, Flavia Delicato, Paulo Pires

Journal of Internet Services and Applications

RESEARCH Open Access

A semantic-based discovery service for the Internet of Things

Porfírio Gomes[1], Everton Cavalcante[1*], Thais Batista[1], Chantal Taconet[2], Denis Conan[2], Sophie Chabridon[2], Flavia C. Delicato[3] and Paulo F. Pires[3]

Abstract

With the Internet of Things (IoT), applications should interact with a huge number of devices and retrieve context data produced by those objects, which have to be discovered and selected a priori. Due to the number, heterogeneity, and dynamicity of resources, discovery services are required to consider many selection criteria, e.g., device capabilities, location, context data type, contextual situations, and quality. In this paper, we describe *QoDisco*, a semantic-based discovery service that addresses this requirement in IoT. *QoDisco* is composed of a set of repositories storing resource descriptions according to an ontology-based information model and it provides multi-attribute and range querying capabilities. We have evaluated different approaches to reduce the inherent cost of semantic search, namely parallel interactions with multiple repositories and publish-subscribe interactions. This paper also reports the results of some performance experiments on *QoDisco* with respect to these approaches to handle resource discovery requests in IoT.

Keywords: Internet of Things, Discovery service, Semantic web, Quality of context, Information model

1 Introduction

The *Internet of Things* (IoT) has rapidly evolved in recent years as an extension of the current Internet with seamlessly interconnected physical objects (a.k.a. IoT devices) towards providing value-added applications for end-users. These devices typically encompass sensors and actuators: sensors produce context data that describe things and their environment whereas actuators are capable of actuating on them. IoT devices provide services that expose functionalities to an IoT ecosystem, but the implementation of those services and their provided interfaces are highly dependent on the underlying device hardware.

With the growth of IoT and the provision of a huge number of devices in the near future, there is a need for *discovery services* to enable clients such as middleware platforms, end-users, and applications to retrieve IoT available resources (i.e., sensors, actuators, services, context data) based on search criteria. Even though discovery services represent a well-studied topic in distributed systems [1], traditional approaches are not suitable for the IoT mainly due to the high dynamics of IoT resources [2]. Existing resource discovery proposals for the IoT usually rely on a single repository responsible for storing resource descriptions [3–6]. There are several well-known drawbacks that make this approach unsuitable for IoT in a real-world scenario, e.g., low scalability and dependability. To overcome these limitations, other approaches have relied on distributed repositories to support the discovery process [7–11].

The high heterogeneity of IoT resources calls for an *information model* containing unambiguous descriptions of IoT resources. Delicato et al. [12] point out that discovering resources and retrieving data in several scenarios of IoT is a challenge itself. This is worsened by the current lack of standardization of protocols and formats to represent resources and the absence of consensus in terms of which concepts should be modeled in an information model for IoT. Chun et al. [13] argue that an information model tailored to IoT should be modeled in terms of *ontologies*, which can represent a set of common concepts used in IoT to model resources/services and relationships among them while providing formal expressiveness, avoiding ambiguity, and fostering automated analysis and inferences. Therefore, enriching information models with this type of information can promote interoperability

*Correspondence: everton@dimap.ufrn.br
[1]Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil
Full list of author information is available at the end of the article

among resources, data providers, and data consumers, and facilitate data access, service integration, semantic interpretation, and knowledge extraction [14].

Discovery services for IoT can significantly benefit from *context-awareness* as resources can be selected according to contextual information, namely the current situation of users and/or applications with minimal user intervention [15]. For instance, a service could search for sensors at a given location nearby a given user or in a room above a certain level of temperature. Another benefit of considering context is that such resources could be prioritized or recommended according to the interest of users as determined by their context, including location, history, etc. Guinard et al. [3] indeed argue that it should be possible to use external sources of information to better formulate queries in IoT, going beyond simple keyword search and using user-quality parameters such as context.

IoT services are deployed in highly dynamic environments in which resources and services constantly degrade, disappear and reappear, etc., thus making data in IoT to be imperfect and inconsistent [12, 16]. A solution to limit such an imperfection is to consider additional knowledge in the form of *Quality of Context* (QoC) metadata, e.g., precision, accuracy, up-to-dateness, etc. [17] to better characterize the acquired data. Using this additional information is relevant since it can augment the richness of the discovery procedure, better support decision-making actions, enable to filter out irrelevant data with insufficient quality, and rank which information should be used [18, 19].

Aiming at addressing the aforementioned concerns, we have developed *QoDisco*, a semantic-based discovery service for the IoT [20]. *QoDisco* is composed of a set of independent repositories storing both sensors and actuators descriptions as well as data produced by sensors. Furthermore, *QoDisco* complies with important requirements for discovery services tailored to IoT [7, 21], such as the ability to handle heterogeneous resources, multi-attribute and range queries, and quality of context data. *QoDisco* encompasses an ontology-based information model with a consistent vocabulary of concepts related to IoT resources, relationships among them, contextual data, and QoC-related information. This information model takes advantage of well-established, standardized ontologies and meta-models to semantically describe resources, services, and QoC criteria, besides easing semantic queries that inherently support multi-attribute and range queries.

Our previous work [20] describes an early version of *QoDisco* by focusing on the presentation of a novel information model to describe IoT resources as well as proposing an initial architecture and implementation prototype. The search process in *QoDisco* has two operation modes, namely synchronous and asynchronous. In this paper, we go a step beyond by bringing improvements in the search process to make it more scalable. We have enriched the *QoDisco* architecture and implementation to support queries over *QoDisco* repositories in parallel, directly improving the time performance of synchronous searches. Moreover, we incorporated in *QoDisco* an internal broker to support notifying clients interested in new records available at repositories, thus removing any dependency from an external service in the asynchronous search process and reducing coupling. We have also performed four computational experiments in an urban pollution monitoring scenario aimed to quantitatively evaluate *QoDisco* and assess the efficiency of its synchronous and asynchronous search processes.

The remainder of this paper is organized as follows. Section 2 describes *QoDisco*, its architecture and implementation. Section 3 presents the performed evaluation. Section 4 discusses related work. Section 5 contains concluding remarks and directions for future work.

2 *QoDisco*

This section presents *QoDisco*, our QoC-aware discovery service for the IoT. Section 2.1 describes the *QoDisco* architecture. Section 2.2 details the synchronous and asynchronous search processes that can be performed using *QoDisco*. Section 2.3 describes the information model used to describe IoT resources, services, and QoC information. Section 2.4 presents a prototype implementation of our proposal. Further information about *QoDisco* is publicly available at http://consiste.dimap.ufrn.br/projects/qodisco/.

2.1 Architecture

QoDisco relies on a collection of repositories to perform discovery tasks. This approach provides several advantages over using a single repository, mainly scalability, fault-tolerance, independent control of repositories, and data distribution [22]. In the context of this work, *QoDisco* clients can be: (i) IoT devices and gateways publishing resources; (ii) applications searching for records, i.e., descriptions of resources, services and context data in repositories; or (iii) record repositories accessible through *QoDisco*. For becoming available through *QoDisco*, each repository must provide operations for both querying and updating records as defined by its *Repository API*.

After receiving a discovery request containing a domain name and a query, *QoDisco* searches for records stored in the repositories pertaining to the domain specified by the client. In this work, a *domain* is an area of knowledge specified through an ontology document, called repository domain ontology (RDO) document. Figure 1 illustrates the modules composing the *QoDisco* architecture, described as follows.

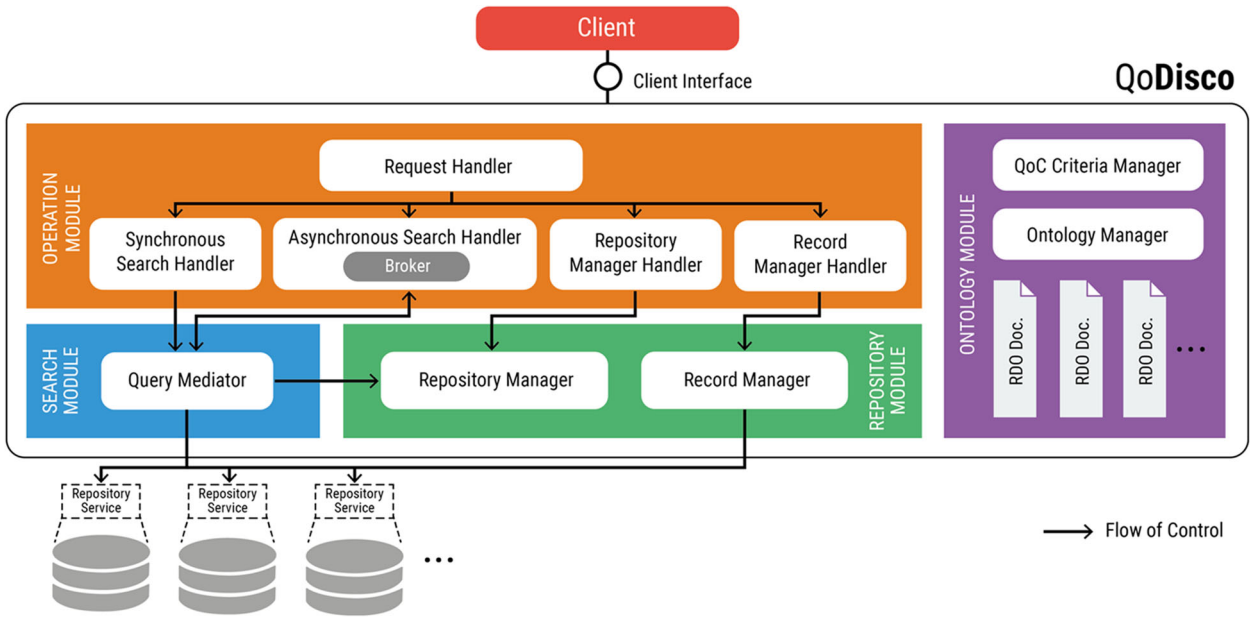

Fig. 1 *QoDisco* architecture

The *Ontology Module* consists of repository domain ontology (RDO) documents describing the concepts composing the information model of *QoDisco*. In this module, the *Ontology Manager* provides operations for adding, removing, and modifying RDO documents, whereas the *QoC Criteria Manager* is responsible for adding, removing, and modifying QoC criteria (also defined in RDO documents). A QoC criterion represents an attribute that qualifies a context information (e.g., accuracy, up-to-dateness) and it can be calculated in different ways according to the situation in which it is used. In *QoDisco*, QoC criteria are described through ontologies to allow for unambiguous identification and representation.

The *Repository Module* manages repositories and maps each one to (a group of) specific RDO document(s) in the *Ontology Module*. As the search and management of records is based on domains, a repository joining *QoDisco* has to be mapped to at least one domain specified by an RDO document and multiple repositories can be mapped to the same domain by their respective owners.

The *Record Manager* is responsible for adding, removing, and modifying records in repositories. As repositories maintain their operational and managerial independence, their owners can manage their own records independently from *QoDisco*. Nonetheless, the information model of *QoDisco* (see Section 2.3) is shared among all of these repositories.

The *Search Module* encompasses the *Query Mediator*, which forwards queries to the repositories belonging to *QoDisco*. To perform the search, this component specifies an RDO document name (i.e., domain name) to the *Repository Manager*, which provides the IP addresses and port numbers of the repositories mapped to the domain.

The *Operation Module* encompasses five components. The *Request Handler* receives requests to search and manage repositories and records for both synchronous and asynchronous operation modes. The *Synchronous Search Handler* performs requests by querying the *Query Mediator*. The *Asynchronous Search Handler* notifies clients (i) about the discovery of a new resource matching the search query and (ii) about the modification or removal of a resource description. The *Repository Manager Handler* and the *Record Manager Handler* respectively interact with the *Repository Manager* and the *Record Manager* to add/remove repositories and records.

The functionalities of the *Operation Module* and the *Ontology Module* are provided through the *Client Interface*. This interface operations regard both synchronous and asynchronous operations as well as the management (insertion, removal, update) of records, repositories, RDO documents, and QoC criteria. More information on the *Client Interface* is also available at http://consiste.dimap.ufrn.br/projects/qodisco/.

2.2 Interaction patterns

Due to the dynamic context where the IoT resources operate, a discovery service in IoT should be able to handle both synchronous calls and asynchronous notifications [23, 24]. The former relies on request-reply interactions towards providing resource information at the moment

of the search whereas the latter are based on publish-subscribe interactions to notify clients in case of resource removal, insertion or update.

The synchronous operation follows the simple request-reply message exchange pattern. This process starts with a client request containing the query and a domain name to be searched for. *QoDisco* identify the repositories that pertain to the specified domain and sends requests to all the matching repositories. Each requested repository searches on its own set of records and returns a response with the set of results. Finally, *QoDisco* combines all results returned by each repository and sends them to the client as a single response. Figure 2 illustrates an example of this process with the *Air Pollution* domain. In this example, only repositories matching the domain name specified by the client are searched by *QoDisco*, i.e., repositories one and three.

The publish-subscribe interaction pattern fosters loose coupling and scalability. In many implementations of this model, information producers (publishers) send messages to a topic from which information consumers (subscribers) interested into such a topic receive all published messages as notifications [25]. This is managed by an intermediary message broker (a.k.a. event bus) that uses a store-and-forward technique to route messages from publishers to subscribers. Figure 3 illustrates this process, which encompasses the following steps:

(1) The client makes a request to the *QoDisco* API with a query and a domain name identifying the records of interest; *QoDisco* stores both query and domain name and sends a response containing the address of the broker within the *Asynchronous Search Handler* and a topic name created to identify the client request;
(2) The client subscribes to the broker by sending the topic name provided by *QoDisco*;
(3) Whenever a record insertion operation is requested, *QoDisco* verifies if such a record meets both query and domain specified by other clients; and
(4) *QoDisco* notifies the new record to interested clients through the broker contained in the *Asynchronous Search Handler*.

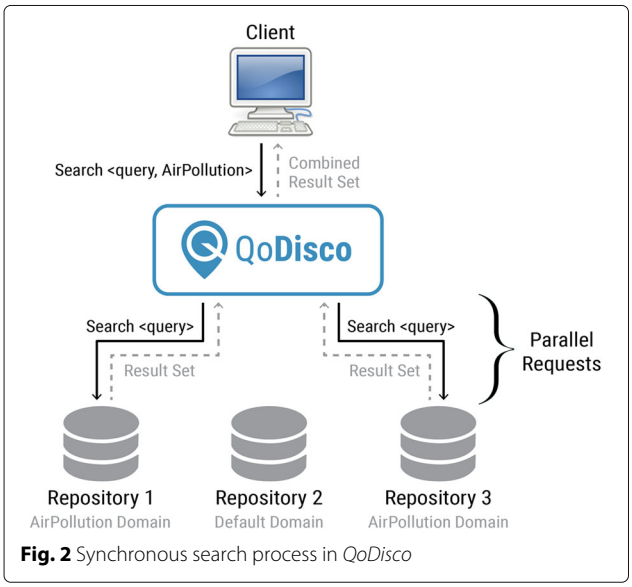

Fig. 2 Synchronous search process in *QoDisco*

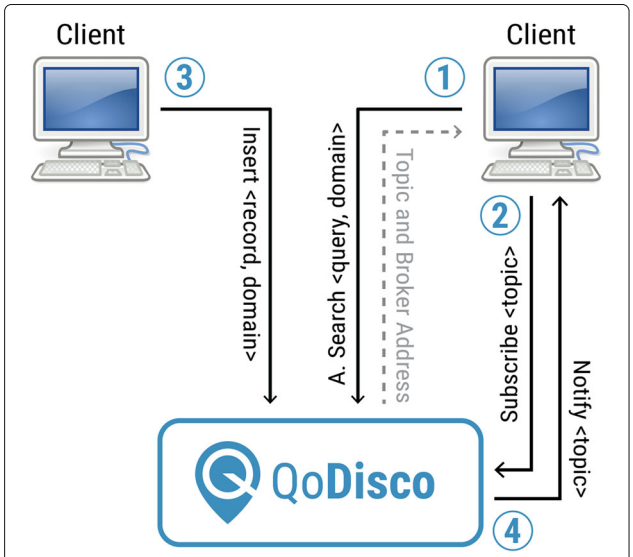

Fig. 3 Asynchronous search process in *QoDisco*

2.3 Information model

The information model of *QoDisco* is depicted in Fig. 4. It comprises an ontology-based vocabulary of concepts related to IoT resources and services, relationships among them, and QoC-related information. To semantically describe resources, such an information model takes advantage of the SAN ontology [26], an extension of the W3C's SSN ontology [27] that provides concepts, attributes, and properties to model both sensors and actuators. Additionally, we have incorporated part of the SOUPA ontology [28] aiming at including location-related concepts to describe location-related information (e.g., latitude, longitude, altitude, distance, surface) as well as symbolic representations of space and spatial relationships, being associated with both resources and data (observations).

The *QoDisco* information model also comprises the OWL-S ontology [29], which has been widely used in both academia and industrial applications for semantically modeling Web services. To represent the relationship between resources and services, we have created the *exposes* property, which links the *Sensor* and *Actuator*

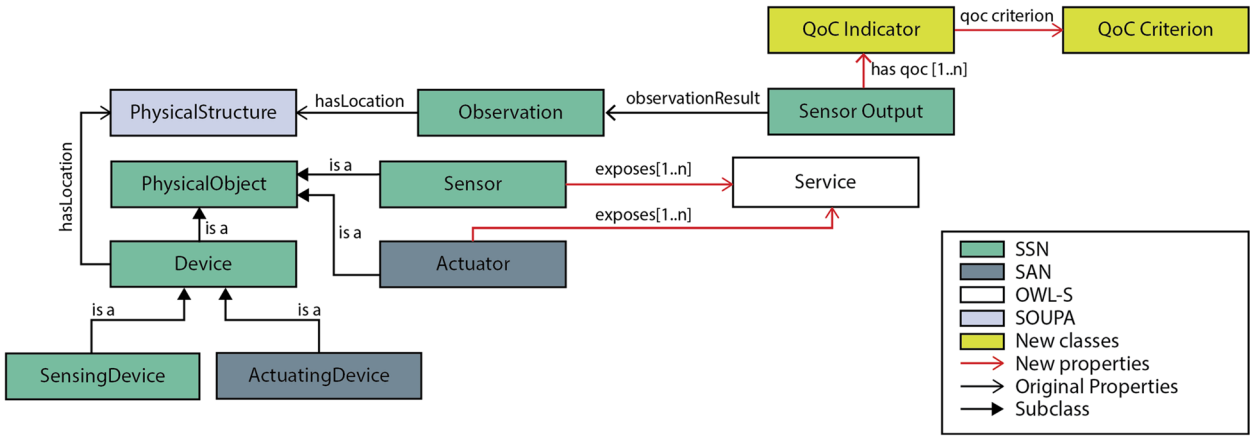

Fig. 4 Information model adopted by *QoDisco*

concepts (representing resources) to the *Service* concept (representing services), as shown in Fig. 4.

To cope with QoC-related concerns, we have incorporated part of the QoCIM meta-model [30] by describing its concepts as an ontology. The *QoC Criterion* concept can be used to represent a QoC parameter associated with context data, e.g., accuracy, precision, completeness, up-to-dateness. In turn, the *QoC Indicator* concept expresses the QoC level of an observation (measured data) made by a resource. In the *QoDisco* information model, the *has_qoc* property associates a QoC level (*QoC Indicator* concept) to an observation made by a resource (*SensorOutput* concept) whereas the *qoc_criterion* property indicates that a QoC indicator must be associated with a QoC criterion. This information model follows the same flexible ideology as QoCIM, i.e., defining a basis to design and represent any QoC criterion instead of providing a predefined list of supported QoC criteria. Moreover, a given QoC criterion can be built upon other primitive or composed QoC criteria.

2.4 Implementation

QoDisco is implemented as a Web application using the Java programming language and the Spring framework, being deployed to an Apache Tomcat server. Spring provides support for multithreading using a thread pool, which was required towards implementing the parallel search request to the repositories (see Section 2.2). For the sake of performance, *QoDisco* uses multithreading facilities to throw searches in parallel.

In the current implementation of *QoDisco*, search queries are specified in SPARQL [31], a semantic query language standardized by W3C for information retrieval. SPARQL supports both multi-attribute and range queries over documents in OWL, the W3C's recommended language to semantically describe ontologies. SPARQL is one of most popular approaches for discovery services due to its relative simplicity of implementation, besides being an open standard. Experimental results reported by Bröring et al. [32] recommend the use of SPARQL for discovery services based on directories (such as *QoDisco*) since it provides queries with richness and result ranking. In particular, search queries in *QoDisco* comply with SPARQL 1.1, the latest version made available by W3C.

The underlying infrastructure for *Operation*, *Search*, *Repository*, and *Ontology Modules* is implemented using Apache Jena [33], an open-source framework to search for, add, and modify OWL descriptions while supporting SPARQL queries. *QoDisco* uses Jena-ARQ as query engine and it follows five steps to perform a query:

(1) *Parsing* – structuration of the query string as query object;
(2) *Algebra Generation* – translation of the query object to an algebra expression using the SPARQL specification algorithm;
(3) *High-level Optimization* – transformations applied to the algebra expression, e.g., replacing equality filters with a more efficient graph pattern;
(4) *Low-level Optimization* – choice of the evaluation order for basic graph patterns; and
(5) *Evaluation* – execution of algebra expressions to generate solution graph patterns.

Information about repositories (HTTP address and supported operations), RDO documents (name and URI), and topics (topic name) is handled by their respective modules to be registered at a relational database.

Repositories are implemented using Apache Fuseki [34], a server providing SPARQL query, update, and storage operations via HTTP requests. Fuseki was chosen due to

its easy deployment and because it provides a SPARQL HTTP endpoint built upon the Jena framework, but other existing technologies could be used to implement repositories. The broker service within the *Asynchronous Search Handler* uses Moquette [35], a lightweight open-source Java library to implement publish-subscribe message brokers for the MQTT messaging protocol. With the broker address and a topic name, a client can interact directly with the Moquette broker in the *Asynchronous Search Handler* using the MQTT protocol.

Finally, the *Client Interface* is implemented as a RESTful Web service using the Spring framework. Clients can access the main functionalities offered by *QoDisco* either through a user-friendly Web graphical interface or through the REST API *Client Interface*.

3 Evaluation

In this section, we report a quantitative evaluation assessing the efficiency of *QoDisco* and its two search processes using computational effort in terms of execution time as metric. Our evaluation does not compare *QoDisco* with other proposals due to the unavailability of benchmarks handling all of its features. Table 1 summarizes the designed experiments.

Section 3.1 describes the scenario considered in the evaluation. Section 3.3 outlines the configuration used in E1 and E2 to assess the performance of the synchronous search process. Section 3.3 presents the configuration used in E3 and E4 to evaluate the performance of the asynchronous search process.

3.1 Scenario

Consider a scenario of air pollution monitoring in an urban area where public buses are equipped with air pollution monitoring sensors and a GPS receiver. Bus stations are equipped with more sophisticated air pollution monitoring sensors capable of providing data with better QoC. Four QoC criteria are used to qualify pollution measurements and GPS location: (i) *margin of error*, which quantifies doubt about the result of a measurement and it is expressed in the same unit as the measurement; (ii) *freshness*, which measures the time elapsed from the collection of observations to its delivery to a consumer; (iii) *precision*, which qualifies how close or how repeatable the results from a measurement are and it is typically expressed as a percentage; and (iv) spatial *resolution*, which measures the precision to express a physical area.

Table 1 Computational experiments for evaluating *QoDisco*

Experiment	Goal
E1	To assess the influence of parallel searches over repositories on the performance of synchronous searches
E2	To assess the performance of *QoDisco* upon an increasing number of records queried in a synchronous search
E3	To assess the performance of the asynchronous search process with a varying number of clients to be notified about the publication of a new record in a repository
E4	To assess the performance of the asynchronous search process to notify a single client with a varying number of records published at the same time

The following snippet illustrates the description of a sensor observation, described as the output of the carbon monoxide sensor *Sensor01*. The observation result is a sensor output referenced as *observation_result01*:

```
qodisco:Observation01
 a ssn:Observation ;
 ssn:observedBy qodisco:Sensor01 ;
 ssn:observedProperty pol:Carbon_Monoxide ;
 ssn:observationResultTime <2018-10-22T16:00:
     24.000-03:00>
 ssn:observationResult :observation_result01 .
```

The value measured by the sensor as well as the QoC criteria associated with it are referenced in *observation_result01* by *observation_qocindicator01* and *observation_resultvalue01*, which respectively are instances of the *QoCIndicator* and *ObservationValue* classes of the *QoDisco* information model:

```
qodisco:observation_result01
 a ssn:SensorOutput ;
 ssn:isProducedBy qodisco:Sensor01 ;
 qodisco:has_qoc qodisco:
     observation_qocindicator01 ;
 ssn:hasValue qodisco:
     observation_resultvalue01 .
```

The following snippet represents a QoC criterion (precision) and its respective value (99%) as referenced in the element *observation_qocindicator01* which is associated to the result observed from the carbon monoxide sensor *Sensor01*:

```
qodisco:observation_qocindicator01
 a qodisco:QoCIndicator ;
 qodisco:has_qoc_criterion qodisco:Precision ;
 qodisco:has_qoc_value 99 .
```

The actual value measured by the carbon monoxide sensor *Sensor01* (397.0) is in turn referenced in the element *observation_resultvalue01*:

```
qodisco:observation_resultvalue01
 a ssn:ObservationValue ;
 ssn:hasQuantityValue 397.0 .
```

In the described scenario, the location of the buses and bus stations is qualified with spatial resolution, pollution measurements provided by buses' sensors are qualified with both precision and freshness, and the ones provided by bus stations' sensors are qualified with margin of error and freshness. These measurements are semantically annotated accordingly to the *QoDisco* information model and stored into a repository registered at *QoDisco*. In this scenario, *QoDisco* can be used by clients to search

for and select sensors and their respective measurements according to their capabilities, location, QoC indicator parameters, etc. using SPARQL queries.

In the experiment, we have calculated the time spent by *QoDisco* to search for all the observations related with carbon monoxide pollution levels with a QoC indicator of precision greater than or equal to 95%. This is expressed by the following SPARQL query:

```
SELECT ?obs ?qocValue ?sensorData WHERE {
  ?obs a ssn:Observation ;
    ssn:observedProperty pol:
        Carbon_Monoxide ;
    ssn:observationResult ?
        observationResult .
    ?observationResult qodisco:has_qoc ?qoc
         ;
    ssn:hasValue ?resultValue .
    ?resultValue ssn:hasQuantityValue ?
        sensorData .
    ?qoc qodisco:has_qoc_criterion qodisco:
        Precision .
    ?qoc qodisco:has_qoc_value ?qocValue .
    FILTER (?qocValue >= 95)
}
```

This query returns a response in JSON or XML containing (i) an URI to the observation, (ii) an URI referring to the sensor output, (iii) a reference to the QoC indicator, and (iv) the QoC value of the observation. Other attributes or additional QoC criteria could be easily added to the search query.

3.2 Experimental setup

As repositories are independently managed by clients and hence they may be deployed to different computational nodes, all experiments used a group of virtual machines over an academic hybrid Cloud Computing service based on the OpenStack platform and VMware Integrated OpenStack. In both E1 and E2, each repository was deployed to an independent virtual single-core machine with 1 GB of RAM using the Fuseki technology [34] whereas *QoDisco* was deployed to a virtual dual-core machine with 2 GB of RAM. A client was deployed to another virtual single-core machine with 1 GB of RAM. These settings are depicted in Fig. 5.

The performance of the synchronous search process in E1 and E2 was assessed by using a JMeter server [36] running in the same virtual machine as the client. Apache JMeter is a well-known tool used to perform load tests and measure performance of Web-based applications, relying on test plans representing a set of actions made by clients. For E1 and E2, a test plan was set with sending an HTTP GET request to the *QoDisco* API with the above-stated SPARQL query.

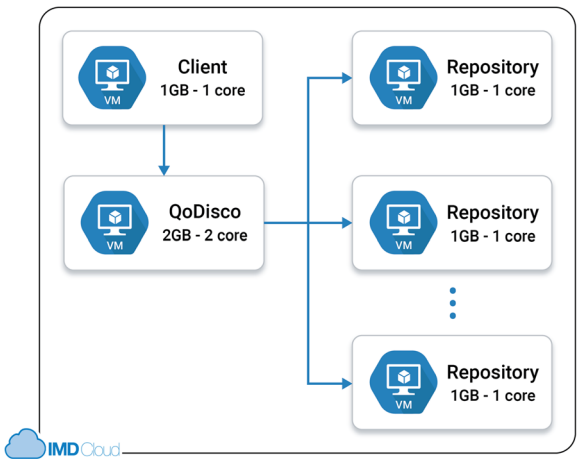

Fig. 5 Computational infrastructure used in experiments E1 and E2 (synchronous search)

In E3 and E4, a repository initially populated with 1,000 random observations related to carbon monoxide pollution levels was deployed in a virtual single-core machine with 1 GB of RAM. *QoDisco* was deployed in a virtual dual-core machine with 2 GB of RAM, multiple publishers were deployed in a virtual single-core machine with 1 GB of RAM, and subscribers were deployed in another single-core machine with 1GB of RAM as well. A time Web service intended to provide the relative time was deployed in a virtual single-core processor with 512 MB of RAM. These settings are depicted in Fig. 6.

As JMeter does not provide native means of subscribing to an MQTT broker, we developed a Java client program to assess the response time of asynchronous notifications sent by *QoDisco* in E3 and E4. To calculate the time spent for the asynchronous search process, the time

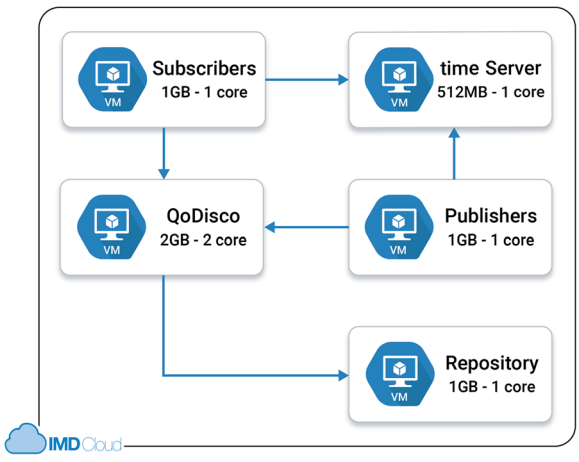

Fig. 6 Computational infrastructure used in experiments E3 and E4 (asynchronous search)

Web service deployed in one of the virtual machines was used to share the current time among clients (publishers and subscribers). This service is accessed to retrieve the current time when publishers request the insertion of new records and subscribers receive notifications from *QoDisco*, so that the elapsed time between these two time instants is determined. We have used such a strategy as distributed systems inherently have no single global notion of the correct time [37]. More importantly than having the correct time is having a consistent time among all computational nodes involved in the experiment and hence a typical practice is using a local time service aiming at minimizing network latency. Therefore, the differences regarding the clocks of the machines used in the experiments is kept as minimal as possible.

An additional, but important caution must be taken when attempting to precisely measure a Java application since it is necessary to consider the complex interaction among the components of the Java Virtual Machine, including the application bytecode, the core runtime system, the just-in-time compiler, and the garbage collector. To obtain a realistic performance measurement, it is necessary to wait for *QoDisco* to reach its steady state after a large number of interactions. In both E1 and E2, we noticed that the synchronous search method reaches a steady state after 1000 requests. For E3 and E4, the steady state was reached after 200 notifications. In all four experiments, we also monitored the processes and threads running in each virtual machine using the `htop` Linux tool [38] to ensure that only *QoDisco* and its Java threads would have considerable amount of time in the dual-core processor.

3.3 Experiments E1 and E2: Synchronous search

In E1, the performance of *QoDisco* is assessed by measuring the time spent to respond to client requests when synchronous searches are performed over repositories in sequence (i.e., one repository after another) and in parallel as the number of available repositories is increased. For this experiment, each repository contained exactly 1,000 randomly generated observations related to carbon monoxide pollution levels, with no other records. Figure 7 presents the results of E1 in terms of average response time (in milliseconds) for the sequential and parallel search approaches of *QoDisco* considering one to five repositories. In this experiment, we measured the performance of *QoDisco* with JMeter by sending a request to the *QoDisco* API from a single thread 50 times. As the number of records returned by the client search could possibly influence the *QoDisco* response time, we have also ensured that exactly 10% of all observations registered in each repository matched the search query.

As shown in Fig. 7, the response time for the synchronous search is improved by using a parallel approach on the available repositories. Average response times in the parallel approach are about 30% smaller than the ones observed in the sequential approach for three repositories and about 40% smaller for five repositories. It is also worth observing that the synchronous search performance of the parallel approach has low variation with the number of repositories, thus revealing the scalability of such an approach with *QoDisco*. In this sense, clients can perform fine-tuned queries by choosing the domains to be searched, thereby reducing the number of repositories being searched and hence improving the performance of *QoDisco*.

In E2, we measured the response time spent by *QoDisco* to respond to a request to the *QoDisco* API from a single thread 50 times, besides ensuring that exactly 10% of the registered observations matched the client query with increasing the number of observations in a single repository. Figure 8 presents the results of E2 in terms of average

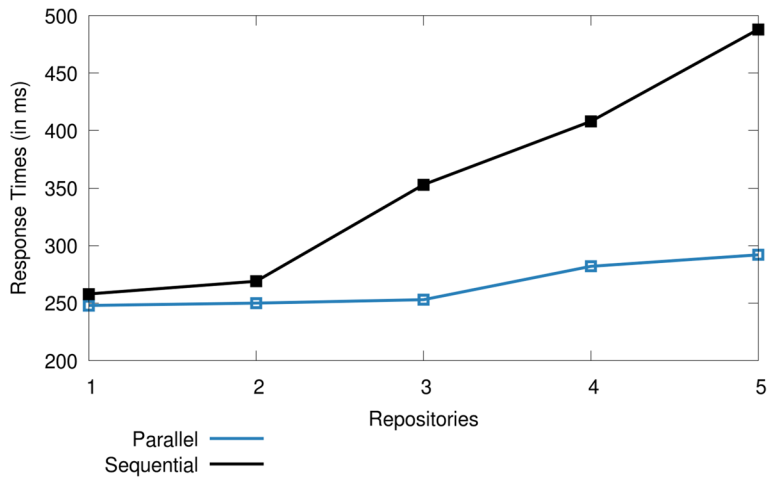

Fig. 7 Results of experiment E1: sequential vs. parallel synchronous search with increasing number of repositories

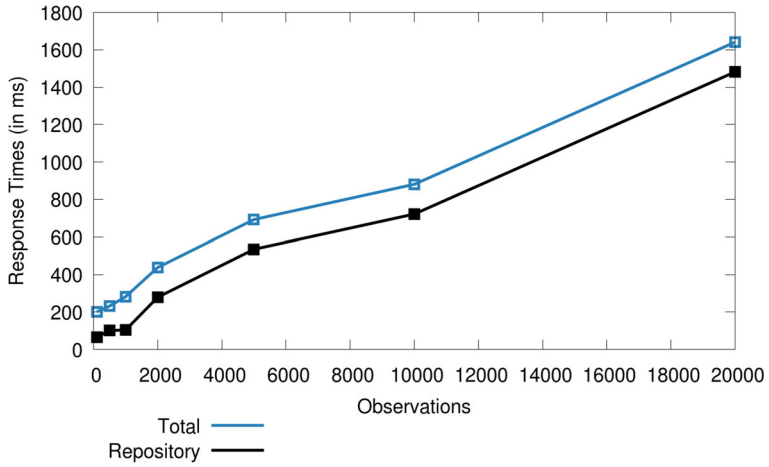

Fig. 8 Results of experiment E2: synchronous search with increasing number of observations

response time (in milliseconds) of the synchronous search process upon increasing the number of registered observations, as well as how much of the total time correspond solely to the repository response time.

Figure 8 shows that the repository response time significantly influences the synchronous search response time. The overhead caused by *QoDisco* is almost constant at 200 ms for all measurements, whereas the repository response time corresponds to approximately 75% of the total response time for 10,000 observations and almost 90% for 20,000 observations. This indicates that the overhead is mainly caused by Apache Fuseki and its storage and query mechanisms.

3.4 Experiments E3 and E4: Asynchronous search

In E3, we performed 50 independent executions of the asynchronous search process while increasing the number of publishers (one to twenty). All publishers were running in the same virtual machines and their publications matched the search query used by the subscriber. The performance was assessed by measuring the time interval from sending the first record to *QoDisco* to the arrival of the last notification to the subscriber. The average response times (in milliseconds) observed in E3 are presented in Fig. 9. The measured time is around 300 ms for 20 simultaneous publications and it scales linearly with the number of publishers matching the query. A proposition to improve the measured time is enforcing a filtering mechanism on *QoDisco* according to a refresh time constraint positioned by the client application, thereby reducing the number of publications to be sent to the client.

To assess the performance of *QoDisco* in E4, 50 independent executions of the asynchronous search process were performed while increasing the number of identical subscribers, one to twenty. The performance was

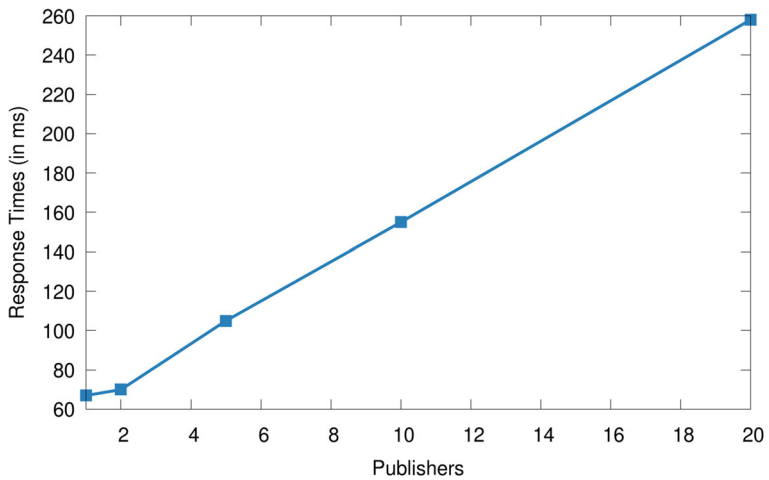

Fig. 9 Results of experiment E3: time to notify a client while increasing the number of publishers

assessed by measuring the time interval from sending a record to *QoDisco* to the notification arrival to all subscribers. Although subscribers were interested in the same records from the same domain (thus representing the worst case in which all subscribers are interested into receiving all available information), we created a topic for each subscriber to simulate independent asynchronous search requests. The average response times (in milliseconds) observed in E4 are shown in Fig. 10 with average response times smaller than 350 ms in the worst experimental condition (20 subscribers). In comparison to our previous work [20], we noticed that the asynchronous search process largely benefits from distributing repositories, *QoDisco*, publishers, subscribers, and the time Web service in different computational nodes instead of using a single one.

An important issue to highlight is that the internal broker service used by *QoDisco* may represent a performance bottleneck with respect to the asynchronous notification process. The overhead to deliver one publication indeed increases with the number of its subscribers. As a future work, we plan to compare new brokers [39] as well as new SPARQL servers according to their throughput and latency towards improving scalability in *QoDisco*.

4 Related work

Discovery services represent a well-studied topic in distributed systems. However, most of the existing approaches cannot be directly applied to IoT as they do not comply with some requirements in this context, as (i) the large scale of IoT scenarios, (ii) resource constraints related to capabilities of devices, and (iii) the high volatility of device network connection [7, 12, 40]. In this section, we summarize some significant approaches found in the literature by analyzing them in terms of search processes and support to context-related information.

4.1 Synchronous and asynchronous search processes

The literature has pointed out SPARQL as one of the most popular approaches for discovery services due to its relative simplicity of implementation and because it is an open standard. Experimental results reported by Bröring et al. [32] recommend the use of SPARQL for discovery services based on directories (such as *QoDisco*) since it provides queries with richness and result ranking. However, some proposals make use of other query methods. SmartSearch [41] is a discovery service that uses the Lucene library for indexing, discovering, and dynamically selecting of resources based on their functional characteristics and non-functional features. *QoDisco* shares the same goal in terms of being easy to integrate to other platforms towards providing discovery capabilities, but its operations are provided through an API built upon SPARQL. Moreover, SmartSearch requires a client library to be installed in the resources/clients to interact with it, a drawback considering the limited nature of the IoT resources.

WOTS2E [42] is a search engine able to discover semantically annotated Linked Data sources (i.e., SPARQL endpoints) and then devices and their services using specialized Web crawlers. The automatic discovery of repositories can increase the number of resources and services available for searching and allow for an easier integration with a resource discovery service. Although WOTS2E seems to be in an initial development stage, the automatic discovery and integration of resources available in the Web might be a large step to the adoption of such a platform in IoT, instead of requiring these endpoints

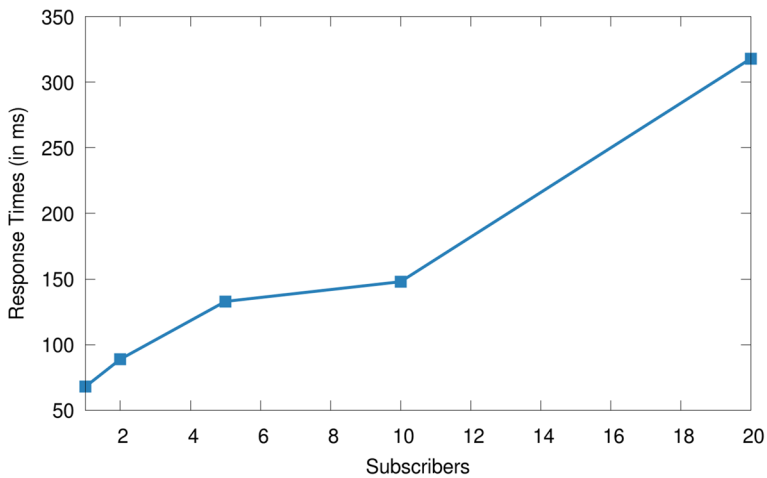

Fig. 10 Results of Experiment 4: time to notify a client while increasing the number of subscribers

to follow a uniform information model and to be manually registered in the platform. As a future work, we consider extending *QoDisco* to support the automatic discovery and integration of SPARQL endpoints containing IoT resources available in the Web.

Another relevant proposal is presented by Chirila et al. [43] with a broker-based architecture for discovery and recommendation of IoT services that supports queries with both functional and non-functional parameters. The broker can cluster based on an initial set of registered services and the registration of new services to the broker can also trigger the dynamic update of the service clusters. When a client queries the platform, the broker firstly performs the search on the clusters based on functional requirements. If multiple Web services meeting the requirements are found, then the selection is further refined based on the quality of the services (non-functional requirements). The strategy adopted by *QoDisco* in terms of dividing repositories into domains also minimizes the discovery cost since only repositories pertaining to the domain specified by the client are searched by the platform.

Cabrera et al. [24] present an evaluation of commonly used service discovery protocols for the IoT, namely DNS-SD, mDNS-SD, CoAP-SD, and DDS Service Discovery. The evaluated protocols provide service description, registration, unregistration, discovery, and resolution capabilities. In their experiments, more than 50% of retrieved services were not relevant to the query used when the services have syntactic differences in their descriptions and more than 60% when the services have semantic differences. Each protocol indeed defines distinct, limited service representations, thus defining a uniform standard to represent IoT resources (such as the *QoDisco* information model) can improve interoperability among IoT protocols and technologies as well as their expressiveness.

We have noticed that the literature has not employed much effort on the asynchronous discovery of resources and notifying clients when there is a new or better resource providing a given service [44]. Many contributions regarding this type of search are built upon the publish-subscribe model. One of them is the Ketema et al.'s work [45], which proposes pushing data directly from specific resources on constrained devices using the CoAP protocol. In short, a client subscribes to event messages produced by a given hosted resource. If the discovery service accepts the request, then the client receives a notification message whenever the resource changes. Although the literature provides some other discovery service proposals, most of them are available as architecture proposals or pilot implementations, thereby indicating there is still a need of a discovery service for IoT supporting an asynchronous search process for semantically annotated resources.

4.2 Using context-related information in discovery

Perera et al. [15] analyze fifty context-aware projects and identify context discovery as one of the six challenges in which new solutions are required in the IoT scenario. The service discovery task can significantly benefit from context-awareness, encompassing understanding sensor data produced by context sources and automatically relating them to high-level context information. Despite this importance and the nature of data in IoT, we have noticed that not much effort has been dedicated to resource discovery based on QoC-related information.

CASSARAM [46] is a centralized solution that uses sensor characteristics and context information such as reliability, accuracy, and battery life registered at a single, centralized registry to automatically search for, select, and rank appropriate sensors among a large set of available devices according to user-defined criteria and priorities. Similarly to *QoDisco*, CASSARAM is built upon an ontology-based information model comprising the SSN ontology and the SPARQL language is used in the search process. Despite CASSRAM addresses context information, it does not consider QoC criteria in the discovery process, which is valuable towards providing more accurate results according to application/user requirements. Indeed, we have not found any proposal in the literature addressing the use of QoC criteria for discovery services in IoT, thereby constituting a significant contribution of our work.

The main motivation of the Wang et al.'s work [47] is to solve two major challenges involved in IoT environments, namely interoperability and scalability. They adopt the sensor-as-a-service vision, in which the ultra-large-scale sensing infrastructure (consisting of wireless sensor networks) provides sensing services for multiple applications and geographically distributed users. In this view, specific devices, their low-level features, and their exact location are abstracted away from users as high-level descriptions provided by a set of standardized interfaces. Although attractive, the authors argue that the sensor-as-a-service vision still lacks the support from a concrete implementation, mainly concerning the requirement of discovering services in an efficient way. They propose a service discovery mechanism based on semantic technologies to deal with heterogeneity and interoperability issues, as well as on a geospatial indexing approach to deal with scalability. As in our proposal, the authors adopt distributed service repositories to deal with scalability and exploit information on spatial context as a way of optimizing queries to repositories, but they give emphasis to geographic features, i.e., the type of context explored is a spatial one. It is not within the scope of their proposal to deal with other types of contextual data nor to consider information about QoC as in our proposal. Another difference with respect to our work is that

the search process is based on the use of the SPARQL semantic query language, which adopts a synchronous interaction model. *QoDisco* provides both synchronous and asynchronous queries, which gives it greater flexibility of operation and support for more diverse types of IoT applications.

Fredj et al. [48] propose a hierarchical-based approach of semantic gateways to improve the discovery of IoT semantic Web services in a dynamic context. In such an approach, the IoT environment is modeled as a tree hierarchy of smart spaces. Each smart space is controlled by a semantic gateway that maintains information about the IoT services in its scope (located within the space) and processes discovery requests. To minimize the discovery cost, the approach proposes creating clusters of similar services that can be optimized over time in terms of number of clusters and number of services per cluster. The discovery cost is measured in terms of the number of service request matching operations performed in a gateway to discover services matching an incoming request. Similarly to *QoDisco*, this approach is based on distributing data structures to support the discovery process. However, the goal is to minimize the discovery cost instead of dealing with scalability issues as is the case in *QoDisco*. Moreover, Fredj et al. [48] use only service location as context information and QoC is not considered in the scope of their proposal.

5 Conclusion

The heterogeneity of the widely distributed plethora of devices in IoT poses a significant challenge to find, select, and use IoT resources (devices, sensors, actuators, services and context data). Therefore, it is important to provide a flexible mechanism that enables clients to easily discover, retrieve, and use device capabilities and data produced by them in an unambiguous way. Due to the imperfect nature of data provided by IoT devices, it is valuable to complement observations with QoC criteria towards augmenting the capability of the search procedure and better supporting decisions made upon these data.

Our research in this context has proposed *QoDisco*, a semantic-based discovery service for IoT structured upon a set of repositories storing resource descriptions and QoC-related information. Unlike many approaches commonly found in the literature, *QoDisco* is rich in the sense that it complies with important requirements for discovery services in IoT, such as searches based on multiple attributes, range queries, different interaction patterns, and QoC criteria. Moreover, *QoDisco* encompasses an information model that takes advantage of well-established, standardized ontologies to semantically describe both resources and services.

The uniform interface to discover resources, a well-defined information model, and richness of the semantic-based querying capabilities provides *QoDisco* with interesting features to handle the inherent heterogeneity observed in IoT. In this paper, we detailed the *QoDisco* architecture and implementation by focusing on improvements aimed to make search processes more scalable and loosely coupled. Obtained experimental results with *QoDisco* have shown that the strategy of distributing repositories and providing asynchronous search process can be promising to resource discovery in IoT.

As future work, we intend to perform additional experiments considering network environments closer to the ones observed in IoT, with low bandwidth, high latencies, lossy network links, etc. We also need to consider other important facets in discovery services for IoT, such as dynamism and security/privacy concerns regarding resources and their metadata, as well as supporting authentication and authorization policies for accessing repositories. In regard to repositories, we aim to investigate other alternatives to RDF query and storage as means of possibly reducing the overhead caused by the mechanisms currently used in *QoDisco*. Furthermore, it is important to provide *QoDisco* with a strategy for repository consistency as repositories do not reflect updates on ontologies, so that repository owners are currently responsible for ensuring such a consistency. Finally, we aim to make insertion and search for records using the *QoDisco* API easier, especially for application developers unfamiliar with the SPARQL language.

Acknowledgements
This work was partially supported by CNPq - Brazilian National Council for Scientific and Technological Development and INES 2.0 - Brazilian National Institute of Science and Technology for Software Engineering (http://www.ines.org.br), funded by FACEPE grant APQ-0399-1.03/17 and CNPq grant 465614/2014-0. TB, FCD and PFP are CNPq fellows.

Funding
Not applicable.

Availability of data and materials
The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.

Authors' contributions
The authors have equally contributed to the manuscript. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil. [2]SAMOVAR-UMR CNRS, Université Paris-Saclay/Télécom SudParis, Évry, France. [3]Department of Computer Science, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil.

Received: 5 September 2018 Accepted: 15 March 2019
Published online: 15 May 2019

References

1. Object Management Group. Trading Object Service Specification Version 1.0. https://www.omg.org/spec/TRADE/About-TRADE/.
2. Aziez M, Benharzallah S, Bennoui H. A comparative analysis of service discovery approaches for the Internet of Things. Int Res J Electron Comput Eng. 2017;3(1):17–22.
3. Guinard D, Trifa V, Karnouskos S, Spiess P, Savio D. Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of Web services. IEEE Trans Serv Comput. 2010;3(3):223–35.
4. Issarny V, Georgantas N, Hachem S, Zarras A, Vassiliadist P, Autili M, et al. Service-oriented middleware for the Future Internet: State of the art and research directions. J Internet Serv Appl. 2011;2(1):23–45.
5. Wei Q, Jin Z. Service discovery for Internet of Things: A context-awareness perspective. In: Proceedings of the Fourth Asia-Pacific Symposium on Internetware. New York: ACM; 2012.
6. Cassar G, Barnaghi P, Wang W, Moessner K. A hybrid semantic matchmaker for IoT services. In: Proceedings of the 2012 IEEE International Conference on Green Computing and Communications. USA: IEEE; 2012. p. 210–6.
7. Paganelli F, Parlanti D. A DHT-based discovery service for the Internet of Things. J Comput Netw Commun. 2012;2012:1–11.
8. Cirani S, Davoli L, Ferrari G, Léone R, Medagliani P, Picone M, et al. A scalable and self-configuring architecture for service discovery in the Internet of Things. IEEE Internet of Things J. 2014;1(5):508–21.
9. Li J, Zaman N, Li H. A decentralized locality-preserving context-aware service discovery framework for Internet of Things. In: Proceedings of the 2015 IEEE International Conference on Services Computing. USA: IEEE; 2015. p. 317–23.
10. Hussein D, Park S, Crespi N. A cognitive context-aware approach for adaptives services provisioning in Social Internet of Things. In: Proceedings of the 2015 IEEE International Conference on Consumer Electronics. USA: IEEE; 2015. p. 192–3.
11. Jo HJ, Kwon JH, Ko IY. Distributed service discovery in mobile IoT environments using Hierarchical Bloom Filters. In: Cimiano P, Frasincar F, Houben GJ, Schwabe D, editors. Proceedings of the 15th International Conference on Engineering the Web in the Big Data Era. vol. 9114 of Lecture Notes in Computer Science. Cham, Switzerland: Springer International Publishing; 2015. p. 498–514.
12. Delicato FC, Pires PF, Batista T. Resource management for the Internet of Things. Cham. Switzerland: Springer International Publishing AG; 2017.
13. Chun S, Seo S, Oh B, Lee KH. Semantic description, discovery and integration for the Internet of Things. USA: IEEE; 2015, pp. 272–5.
14. Wang W, De S, Toenjes R, Reetz E, Moessner K. A comprehensive ontology for knowledge representation in the Internet of Things. USA: IEEE; 2012, pp. 1793–8.
15. Perera C, Zaslavsky A, Christen P, Georgakopoulos D. Context aware computing for the Internet of Things: A survey. IEEE Commun Surv Tutor. 2014;16(1):414–54.
16. Henricksen K, Indulska J. Modelling and using imperfect context information. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops. USA: IEEE; 2004. p. 33–37.
17. Buchholz T, Küpper A, Schiffers M. Quality of Context: What it is and why we need it. In: Proceedings of the 10th International Workshop of the OpenView University Association. Geneva; 2003. p. 1–14.
18. Marie P, Desprats T, Chabridon S, Sibilla M. Extending Ambient Intelligence to the Internet of Things: New challenges for QoC management. In: Hervás R, Lee S, Nugent C, Bravo J, editors. Proceedings of the 8th International Conference on Ubiquitous Computing and Ambient Intelligence. vol. 8867 of Lecture Notes in Computer Science. Cham, Switzerland: Springer International Publishing; 2014. p. 224–31.
19. Chabridon S, Laborde R, Desprats T, Oglaza A, Marie P, Marquez SM. A survey on addressing privacy together with quality of context for context management in the Internet of Things. Annals Telecommun. 2014;69(1): 47–62.
20. Gomes P, Cavalcante E, Batista T, Taconet C, Chabridon S, Conan D, et al. In: García CR, Caballero-Gil P, Burmester M, Quesada-Arencibia A, editors. A QoC-aware discovery service for the Internet of Things. Cham, Switzerland: Springer International Publishing; 2016, pp. 344–55.
21. Gomes P, Cavalcante E, Rodrigues T, Batista T, Delicato FC, Pires PF. A federated discovery service for the Internet of Things. In: Proceedings of the 2nd Workshop on Middleware for Context-Aware Applications in the IoT. New York: ACM; 2015. p. 25–30.
22. Schmidt C, Parashar M. A peer-to-peer approach to Web service discovery. World Wide Web. 2004;7(2):211–29.
23. Brambilla M, Umuhoza E, Acerbis R. Model-driven development of user interfaces for IoT systems via domain-specific components and patterns. J Internet Serv Appl. 2017;8:.
24. Cabrera C, Palade A, Clarke S. An evaluation of service discovery protocols in the Internet of Things. New York: ACM; 2017, pp. 469–76.
25. Eugster PT, Felber PA, Guerraoui R, Kermarrec AM. The many faces of publish/subscribe. ACM Comput Surv. 2003;35(2):114–31.
26. Spalazzi L, Taccari G, Bernardini A. An Internet of Things ontology for earthquake emergency evaluation and response. In: Proceedings of the 2014 International Conference on Collaboration Technologies and Systems. USA: IEEE; 2014. p. 528–34.
27. Barnaghi P, et al. Semantic Sensor Network XG Final Report. http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/. Accessed Dec 2018.
28. Chen H, Finin T, Joshi A. The SOUPA ontology for Pervasive Computing. In: Tamma V, Cranefield S, Finin TW, Willmott S, editors. Ontologies for agents: Theory and experiences. Whitestein Series in Software Agent Technologies. Birkhäuser Basel: Cham, Switzerland; 2005. p. 233–58.
29. Martin D, et al. Bringing semantics to Web services: The OWL-S approach. In: Cardoso J, Sheth A, editors. Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition. vol. 3387 of Lecture Notes in Computer Science. Germany: Springer Berlin Heidelberg; 2005. p. 26–42.
30. Marie P, Desprats T, Chabridon S, Sibilla M. The QoCIM Framework: Concepts and tools for Quality of Context management. In: Brézillon P, Gonzalez AJ, editors. Context in Computing: A cross-disciplinary approach for modeling the real world. USA: Springer New York; 2014. p. 155–72.
31. World Wide Web Consortium (W3C). SPARQL Query Language for RDF. https://www.w3.org/TR/sparql11-overview//. Accessed Dec 2018.
32. Bröring A, Datta SK, Bonnet C. A categorization of discovery technologies for the Internet of Things. In: Proceedings of the 6th International Conference on the Internet of Things. New York: ACM; 2016. p. 131–9.
33. Apache Software Foundation. Jena: A free and open source Java framework for building Semantic Web and Linked Data applications. https://jena.apache.org/. Accessed Dec 2018.
34. Apache Software Foundation. Apache Jena Fuseki. https://jena.apache.org/documentation/fuseki2/. Accessed Dec 2018.
35. Selva A. Moquette MQTT broker. http://moquette.io/. Accessed Dec 2018.
36. Apache Software Foundation. Apache JMeter. http://jmeter.apache.org/. Accessed Dec 2018.
37. Coulouris G, Dollimore J, Kindberg T, Blair G. Distributed systems: Concepts and design, 5th ed. Boston: Addison-Wesley/Pearson Education, Inc.; 2012.
38. Muhammad H. `htop` - An interactive process viewer for Unix. http://hisham.hm/htop/. Accessed Dec 2018.
39. Sommer P, Schellroth F, Fischer MT, Schlechtendahl J. Message-oriented middleware for industrial production systems. In: Proceedings of the 14th IEEE International Conference on Automation Science and Engineering. USA: IEEE; 2018. p. 1217–23.
40. Issarny V, Bouloukakis G, Georgantas N, Billet B. Revisiting Service-Oriented Architecture for the IoT: A middleware perspective. In: Sheng QZ, Stroulia E, Tata S, Bhiri S, editors. Proceedings of the 14th International Conference on Service-Oriented Computing. vol. 9936 of Lecture Notes in Computer Science. Cham, Switzerland: Springer International Publishing; 2016. p. 3–17.
41. Fortino G, Lackovic M, Russo W, Trunfio P. A discovery service for smart objects over an agent-based middleware. In: Pathan M, Wei G, Fortino G, editors. Proceedings of the 6th International Conference on Internet and Distributed Computing Systems. vol. 8223 of Lecture Notes in Computer Science. Germany: Springer Berlin Heidelberg; 2013. p. 281–93.
42. Kamilaris A, Yumusak S, Ali MI. WOTS2E: A search engine for a Semantic Web of Things. In: Proceedings of the 3rd IEEE World Forum on Internet of Things. USA: IEEE; 2016. p. 436–41.
43. Chirila S, Lemnaru C, Dinsoreanu M. Semantic-based IoT device discovery and recommendation mechanism. In: Proceedings of the 12th International Conference on Intelligent Computer Communication and Processing. USA: IEEE; 2016. p. 111–6.

44. Vandana CP, Chikkamannur AA. Study of resource discovery trends in Internet of Things (IoT). Int J Adv Netw Appl. 2016;8(3):3084–9.
45. Ketema G, Hoebeke J, Moerman I, Demeester P, Tao LS, Jara AJ. Efficiently observing Internet of Things resources. In: Proceedings of the 2012 IEEE International Conference on Green Computing and Communications. USA: IEEE; 2012. p. 446–9.
46. Perera C, Zaslavsky A, Christen P, Compton M, search GeorgakopoulosD. Context-awaresensor. selection and ranking model for Internet of Things middleware. In: Proceedings of the 14th IEEE International Conference on Mobile Data Management. USA: IEEE; 2013. p. 314–22.
47. Wang W, Deb S, Cassarc G, Moessner K. An experimental study on geospatial indexing for sensor service discovery. Expert Syst Appl. 2015;42(7):3528–38.
48. Fredj SB, Boussard M, Kofman D, Noirie L. Efficient semantic-based IoT service discovery mechanism for dynamic environments. In: Proceedings of the 25th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communication. USA: IEEE; 2014. p. 2088–92.