C Minor: a Semantic Publish/Subscribe Broker for the Internet of Musical Things

Fabio Viola* ✉ , Luca Turchet†, Francesco Antoniazzi*‡ and György Fazekas† ✉
*University of Bologna, Bologna, Italy
Email: {name.surname}@unibo.it
†Queen Mary University of London, London, United Kingdom
Email: {luca.turchet,g.fazekas}@qmul.ac.uk
‡Centro Nazionale Tecnologie Informatiche e Telematiche (INFN CNAF), Bologna, Italy

***Abstract*—Semantic Web technologies are increasingly used in the Internet of Things due to their intrinsic propensity to foster interoperability among heterogenous devices and services. However, some of the IoT application domains have strict requirements in terms of timeliness of the exchanged messages, latency and support for constrained devices. An example of these domains is represented by the emerging area of the Internet of Musical Things. In this paper we propose C Minor, a CoAP-based semantic publish/subscribe broker specifically designed to meet the requirements of Internet of Musical Things applications, but relevant for any IoT scenario. We assess its validity through a practical use case.**

Keywords*—*Internet of Musical Things; Smart musical instruments; Smart spaces; Semantic Web; Context-aware computing.

I. Introduction

In the end of the eighties, a visionary sentence by Mark Weiser [1] put the basis for the current Internet of Things (IoT), definition appeared for the first time only ten years later [2]. Weiser stated that "*the most profound technologies are those that disappear*" [1] and this is very close to what the IoT is building around us. Many are the application domains where the IoT is employed and several are the steps that brought to the Internet of Things (i.e., pervasive or ubiquitous computing [3], context-aware computing [4]), but one thing remained unchanged: the central role of the context. Abowd et al. provided a definition of context that is the one most commonly accepted in literature: "*context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*" [4]. The Internet of Things is gradually evolving towards a new direction known as Semantic Web of Things [5], where the main focus is on the adoption of technologies belonging to the area of Semantic Web [6]. The Semantic Web was born to transform the Web from a repository of human-readable information into an entity that allows for the machine-understandability of data. This vision is becoming reality thanks to a layered structure known as Semantic Web stack. The main components of this stack allow: 1) the univocal identification of resources thanks to IRI (*International Resource Identifiers*); 2) the representation of data as a set of triples thanks to RDF (*Resource Description Framework*) [7]; 3) the definition of a vocabulary to clearly state the meaning of represented data by the means of RDFS (*RDF Schema*) [8] and OWL (*Web Ontology Language*) [9]; 4) the storage and retrieval of data via the SPARQL Update [10] and Query languages [11].

Internet of Things and Semantic Web evolved independently, and the various attempts to combine the strengths of the latter (e.g., simple data representation formalism, machine-understandability and disambiguation of meaning) with the requirements of the IoT (e.g., scalability, timeliness and support for constrained devices) revealed themselves to be ambitious tasks, which are still ongoing. In particular, the biggest issue of today's IoT is well resumed by the word *fragmentation*. The IoT scenario, in fact, is currently characterized by high heterogeneity of devices, services and protocols, which is an issue limiting the interoperability [12]. Semantic Web standards allow one to rely on a common representation of data based on shared and agreed ontologies, and provide a powerful mechanism to solve the problems of discoverability and orchestration of services [13]. Among the hundreds of communication protocols with different aims, lightweigth protocols for machine-to-machine (M2M) communication (e.g., CoAP, MQTT, AMQP) are gaining momentum for their ability to support constrained devices thanks to a binary representation of messages, a small code footprint, and the implementation of paradigms such as publish/subscribe or resource/observe [14], [12]. This allows one to build reactive applications characterized by a constant evolution of the context. However, the Semantic Web stack is oriented towards more static scenarios, where information evolves at a lower rate [15], making challenging bridging these two worlds.

In this paper we exploit a lightweight IoT protocol for M2M communication, the Constrained Application Protocol (CoAP) [16], to build a semantic context broker, named *C Minor*, which is suitable for fog computing applications involving constrained devices [17]. C Minor leverages the authors' experience on the development of semantic publish/subscribe brokers (e.g., for the Smart-M3 interoperability platform [18], [19] and the SPARQL Event Processing Architecture [15]) for context-aware and IoT applications [20], [21]. The proposed approach combines the expressive power of Semantic Web technologies (RDF, RDFS, OWL and SPARQL) with the advantages of a top-class IoT protocol such as CoAP (binary data sent over UDP, resource/observe interaction pattern). We argue that this approach puts the basis for the efficient adoption of semantics on constrained devices while still granting backward compatibility with the old SPARQL 1.1 protocol.

To assess the validity of our approach, we propose an

evaluation of C Minor and a proof of concept based on one of the most challenging IoT scenarios: the Internet of Musical Things (IoMusT) [22]. According to the proposal of Turchet and colleagues, the IoMusT relates to the network of physical objects (*Musical Things*) dedicated to the production, interaction with or experience of musical content. Musical Things embed electronics, sensors, data forwarding and processing software, and network connectivity enabling the collection and exchange of data for musical purpose. A Musical Thing can take the form of a smart musical instrument [23], or any other smart device utilized to control, generate, or track responses to music content. The technology proposed here may be proved particularly useful in IoMusT scenarios where several actors are involved in the music creation process, such as audience members using large-scale participatory live music systems. In such scenarios it is indeed essential to reduce the transmission bandwidth consumption.

The remainder of the paper is organized as follows: in Section II we propose an overview of the state of the art. In Section III the architecture of C Minor is presented, while Section IV contains the results of a preliminary evaluation. Conclusions and future works are discussed in Section V.

II. Related Work

The Internet of Things is a research area that is currently being applied to many application domains with very different requirements. Among these, the Internet of Musical Things [22] is one of the most constrained in terms of latency. Nevertheless, it could benefit from the adoption of technologies belonging to the Semantic Web, for their intrinsic ability to foster interoperability among devices and applications [24]. In this Section, we propose an overview of the state of the art in the area of semantics applied to the IoT and an overview of the main IoMusT applications.

A. Semantic technologies in the IoT

Polling mechanisms are not efficient either for the required computational effort to periodically compare the results of a query with its previous execution, or for the amount of data transferred over the network. A publish/subscribe paradigm [14] allows one to solve these issues: clients communicate their interests once and receive notifications when something matching them is available. Semantic Web technologies are currently bound to a request-response interaction paradigm. An effort toward a publish/subscribe enhanced version of Semantic Web technologies has been made by several researchers, like Murth et al. [25], Alti et al. [26] and Schade et al. [27]. Adapting Semantic Web technologies to more dynamic scenarios is a task carried on also by the Smart-M3 community, which proposes a broker-centric publish/subscribe architecture pivoting the SPARQL language. In [28], the authors reported a survey of the semantic information brokers implemented according to the M3 paradigm. RedSIB [29], the OSGi SIB [19], cuteSIB [30] and pySIB [18] are four of the most recent contributions of the Smart-M3 community to the development of a semantic publish/subscribe architecture for context-aware applications. A new project originated from the research on the Smart-M3 platform: the SPARQL Event Processing Architecture (SEPA) is a publish/subscribe context-broker built on top of a SPARQL endpoint that offers support for standard protocols. Despite being oriented towards IoT applications, these semantic brokers still do not employ the lightweight protocols designed for such scenarios. In literature there are very recent attempts of bridging the gap between semantic technologies and the IoT, but they still neglect the ability to subscribe to changes in the knowledge base (e.g., [31]). In [32], authors focus on the provision of more effective resource discovery, supporting also approximate matches.

Other studies that rely on Semantic Web to build up an IoT environment usually focus on its capability to store uniformly formatted knowledge more than on the dispatching aspect. So, in works like the previously mentioned [24], the Semantic Web is considered as a possible solution to the untreatable verticality of IoT. Data representation is also the topic of interest in [33], where the relationship between Semantic resources and RESTful resources is investigated. Mainetti et al. in [34] developed a parallelism between the MQTT topic concept and Semantic URI resources to achieve the so-called *Web of Topics*. Another work by Khodadadi et al., [35], proposes a framework joining semantics, linked data and a widely used protocol like CoAP targeting with a practical example the collaboration of those technologies, resulting in the possibility of environmental discovery and full usage.

B. Technology-mediated audience participation

As mentioned in Section I, a relevant IoMusT application of the proposed technology concerns systems that involve communications between several actors, such as large-scale audiences. Technology-Mediated Audience Participation (TMAP) [36] is a branch of interactive arts where audience members are actively engaged in the music creation process by means of information and communication technologies. Reviews of TMAP systems can be found in [37], [38], [39], [40]. These systems scatter the conventional unidirectional chain of musical communication for written Western music, where the musical messages are exchanged sequentially from composers to performers to "passive" listeners, since the audience members collaboratively contribute to the music making thus playing also the role of composer and performer. Interaction techniques for TMAP have been proposed exploiting different types of media and sensors, including mobile devices [37], [41], [42], [39] and tangible interfaces [43] such as light sticks [44].

C. Existing IoMusT ecosystems

An IoMusT ecosystem is composed of actors involved in musical activities such as performers and audiences, as well as information and service providers [45]. As for any IoT ecosystem, an IoMusT one forms around commonly used hardware and software platforms as well as standards. From the technological perspective, the core components of an IoMusT ecosystem are of three types: 1) Musical Things (i.e., the interoperable devices serving musical purposes); 2) connectivity (i.e., the network infrastructure supporting multi-directional wireless communication between Musical Things), 3) applications and services (i.e., the enablers for different types of interactions between Musical Things users).

To date, only a handful of IoMusT ecosystems exist, some of which have been devised for ubiquitous music scenarios [46]. For instance, research has investigated technologically-mediated interactions between a performer playing a smart

musical instrument and audience members using Musical Haptic Wearables (MHWs). Smart Instruments are a family of musical instruments characterized by embedded computational intelligence, wireless connectivity, an embedded sound delivery system, and an onboard system for feedback to the player [23], [47]. They offer direct point-to-point communication between each other and other portable sensor-enabled devices connected to local networks and to the Internet. MHWs are wearable devices for audience members, which encompass haptic stimulation, gesture tracking, and wireless connectivity features [48]. The work reported in [49] presents an IoMusT architecture enabling the multidirectional creative communication between a performer playing a smart mandolin and four audience members using armband-based MHWs. The smart mandolin was configured to extract in real-time from the acoustic signal captured by the microphone, the strums performed on the strings. The timing and amplitude of strums having an amplitude above a certain threshold were sent synchronously and simultaneously to four MHWs. Upon reception, such information was mapped to vibrations of amplitude and duration proportional to the detected strum. In addition, to involve the audience in the music making process, the sensor interface of the MHWs were used by audience members to deliver to the smart mandolin player messages activating the playback of backing tracks.

Another example of IoMusT ecosystem is reported in [50]. The authors presented an IoMusT ecosystem involving a smart mandolin, smartphones, and the Freesound repository (https://freesound.org/). The ecosystem was devised to support performer-audience interactions and aimed to enable the creative use of content retrieved from Freesound as an accompaniment for the music played by the smart mandolin. The ecosystem was tested with three audience members, who were empowered to retrieve and organize Freesound content to collaboratively create the accompaniment.

The recent work reported in [51] describes a semantically-enriched Internet of Musical Things architecture which relies on a semantic audio server and edge computing techniques. Specifically, a SPARQL Event Processing Architecture is employed as an interoperability enabler allowing multiple heterogeneous Musical Things to cooperate. The authors created an IoMusT ecosystem around such architecture, which involved basic prototypes of Musical Things.

III. C Minor

To the best of our knowledge, C Minor is the first attempt to exploit CoAP in a SPARQL Event Processing Architecture. A SEPA is a semantic client/server architecture where clients communicate by means of messages exchanged through the server (i.e., also named broker). This broker-centric architecture is then classifiable as a message-oriented middleware [52]. The main assignment of the broker is to hold and provide access to an RDF knowledge base (KB) by means of the SPARQL query and update language: the broker is then a SPARQL endpoint and as such, clients communicate through SPARQL requests. Differently from a standard SPARQL endpoint, a SEPA also implements the publish/subscribe paradigm: subscriptions allow clients to be notified about changes in the KB (i.e., in any subgraph of interest) avoiding polling. C Minor should then provide the ability to retrieve data from a knowledge base through the request/response or publish/subscribe paradigm and update the knowledge base by adding, modifying or deleting triples from it.

Developing a semantic context broker relying on IoT lightweight protocols required an analysis of the available alternatives (i.e., CoAP, MQTT and AMQP). The choice of CoAP was determined by several factors:

- It was proposed by the Internet Engineering Task Force (IETF), and in particular the Constrained RESTful Environments (CoRE) subgroup, as a standard Request For Comment (RFC). So it is an open, fully documented specification [16] and it can paves the way towards interoperability in the IoT.
- CoAP headers have a minimum impact on the message size, [12];
- CoAP is based on UDP (while MQTT and AMQP rely on TCP) but still supports retransmission of lost or damaged packets through the mechanism of confirmable and non confirmable messages. In this way CoAP removes the overhead caused by the three-way handshake protocol;
- As highlighted by Naik [12], CoAP, compared to MQTT and AMQP, presents the lowest bandwidth requirement and the lowest latency;
- CoAP addresses the problem of discoverability [13] by providing a list of the available resources.
- CoAP is designed to be easily mapped on HTTP, then binding the SPARQL 1.1 Protocol [53] to CoAP is a very straight-forward process.

A. From HTTP to CoAP

The SPARQL 1.1 protocol [53] relies on HTTP to convey requests and responses. CoAP can be easily translated to HTTP since it implements a subset of REST optimized for M2M computation [16]. For this reason we report in this Section a comparison between the SPARQL 1.1 Protocol and the CoAP version proposed in this paper:

- SPARQL Update requests: according to [53], a SPARQL Update is sent with an HTTP POST request where the text of the update is specified in the request payload or through url-encoded parameters. A successful request may return a `2XX` or `3XX` code, while a failure is signaled by a `4XX` (for wrong requests, e.g., syntax error) or `5XX` code (server issues). C Minor accepts requests including the text of the update as a payload and returns `2.04 Changed` in case of success, otherwise a `4.00 Bad Request` for a wrong request or `5.XX` code to notify problems on the server side.
- SPARQL Query requests: according to [53], queries can be performed with GET or POST requests. In the first case, the query is specified through percent-encoded parameters. In the latter, as in SPARQL Updates, queries can be provided as a payload or through url-encoded parameters. C Minor accepts

queries provided as payload of POST requests. The status code can be `2.04` (in case of success), `4.00` (for wrong requests) or `5.XX` (for server-side errors).

The comparison between the SPARQL 1.1 Protocol and the CoAP version proposed by C Minor is reported in Table I.

TABLE I. MAPPING SPARQL 1.1 PROTOCOL OVER COAP. A SUMMARY OF THE IMPLEMENTATION PROPOSED IN C MINOR

	HTTP	**CoAP**
Update Request verb	`POST`	`POST`
Text specified as:	payload or url-encoded	payload
Status code for success:	`2XX` or `3XX`	`2.04`
Status code for error:	`4XX` or `5XX`	`4.00` or `5.XX`
Query Request verb	`GET`,`POST`	`POST`
Text specified as:	url-encoded or payload	payload
Status code for success:	`2XX` or `3XX`	`2.04`
Status code for error:	`4XX` or `5XX`	`4.00` or `5.XX`

B. Software architecture

C Minor is a python3 server built on top of the `aiocoap` framework, a natively asynchronous implementation of a CoAP library. C Minor can either exploit the `rdflib` to implement an internal RDF graph (useful to reduce the dependencies on the target device), or rely on an external SPARQL endpoint (e.g., the Fuseki endpoint developed in the context of the Apache Jena framework).

The architecture, depicted in Fig 1, is based on a set of classes, among which is worth mentioning:

- `SPARQLQueryResource` that handles all the requests for SPARQL queries;
- `SPARQLUpdateResource`, responsible for the update of the knowledge base and the subsequent triggering of the subscriptions;
- `SPARQLSubscribeResource`, which is the class that deals with the requests for new subscriptions as well as the requests to close existing ones. Every time a new subscription request is received, a proper instance of the class `SubscriptionResource` is created.
- `SubscriptionResource` is an observable class that permits clients to receive notifications related to a given subscription.
- `CMinorStats`, in charge of collecting stats to analyze the state of the system.
- `Endpoint`, responsible of all the interactions with either the external SPARQL endpoint or the internal graph.

Fig. 1 shows a UML class diagram where, for the sake of clarity, only the relationships among classes defined in the `aiocoap` framework and classes implemented in C Minor are highlighted. Arrows with white head represent an inheritance relationship.

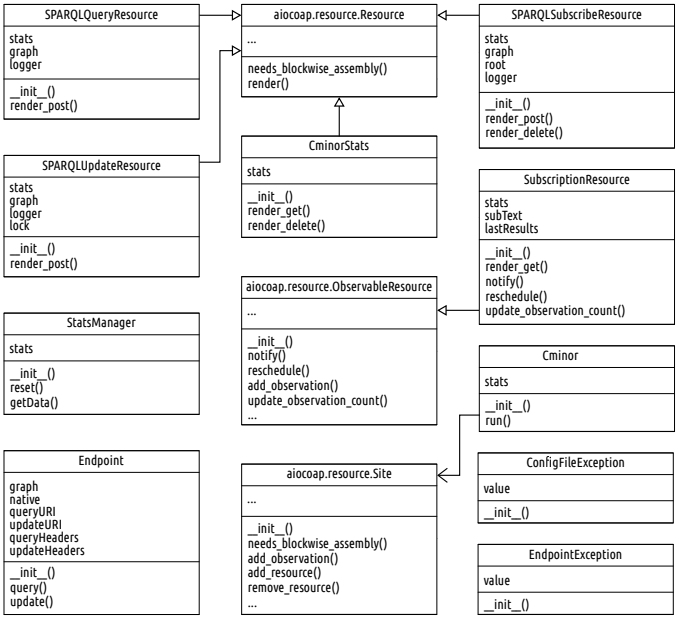

Fig. 1. Class Diagram showing the relationship between C Minor and aiocoap classes

C. Primitives

C Minor holds or interacts with an RDF graph and provides all the functionalities of a SPARQL Endpoint over CoAP. As previously mentioned, following the example of the Smart-M3 platform [28] and the SPARQL Event Processing Architecture [15], the ability to subscribe to a subgraph is provided. The following is a list of all the primitives that should be implemented by client-side libraries to interact with a C Minor instance, in relation to the routes exposed by the server:

- **update** – C Minor, just like SEPA, exposes a proper route (i.e., `/update`) to handle update requests. Then, the only difference with SEPA is represented by the protocol that is CoAP instead of HTTP, but with the same verb (i.e., `POST`, see Section III-A). An UML sequence diagram showing how the user can perform a SPARQL Update is shown in Fig. 2.
- **query** – Queries, just like updates, are handled as in the SPARQL Event Processing Architecture. A `POST` in a CoAP (instead of HTTP) request is used to deliver the query to the `/query` route of the server. Fig 3 depicts the process to perform a SPARQL query.
- **regSubscription** – every subscription corresponds to an observable resource. This primitive is needed to allow the creation of such resource. For example, a `POST` request with the following payload:

```
{
  "query":"SELECT * WHERE { ?s ?p ?o }",
  "alias":"all"
}
```

creates a new observable resource with URI `/all` that provides notifications every time the results of the query changes (i.e., every time the underlying knowledge base is updated). Differently from SEPA, this mechanism allows C Minor to intrinsically group

all the equivalent subscriptions to provide a more efficient management both in terms of computational and memory resources.

- **observe** – client-side libraries should provide the ability to observe resources to exploit the subscription functionality provided by C Minor. A `GET` request with the `observe` option set is what it is needed to start receiving notifications. The URI to observe a resource is the one specified during the `POST` request to `/subscription` (this process is shown in Fig. 4). If the client willing to observe a resource is different from the one that initialized the subscription, then the URI can be discovered through the proper primitive `discovery` described below.
- **unregSubscription** – As the name suggests, this primitive would allow one to unregister a subscription (i.e., to delete the related observable resource). This primitive would perform a `DELETE` request to the route `/subscription` with a payload like this:

```
{ "alias":"all" }
```

- **discover** – C Minor, being based on the CoAP protocol, exposes the resource `/.well-known/core` that enables the discovery on the available resources. A discovery allows one to get the URIs of the routes to update and query the knowledge base, as well as the routes to register or unregister a new subscription. More importantly, the discovery allows to get the list of the observable resources corresponding to active subscriptions, as shown by Fig. 5
- **stats** – C Minor holds an internal data structure containing an updated count of the performed updates and queries with the average elapsed time. Moreover it maintains a list of the active subscriptions, with the number of generated notifications. These statistics can be retrieved with a `GET` request on the `/stats` URI, or reset with a `DELETE` on the same URI. Supporting the access to statistics in client-side APIs is optional.

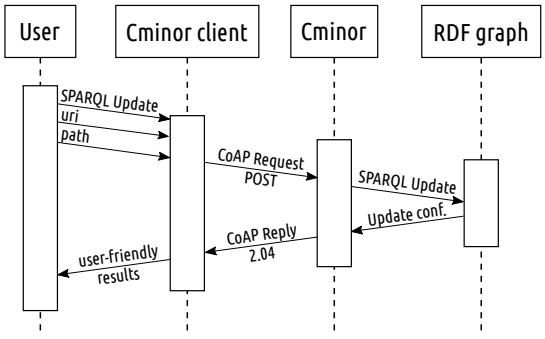

Fig. 2. Sequence diagram for SPARQL updates

A list of the URIs exposed by the C Minor server is proposed in Table II.

IV. Evaluation

In this Section we propose the results of a preliminary evaluation of C Minor in order to characterize the behaviour of

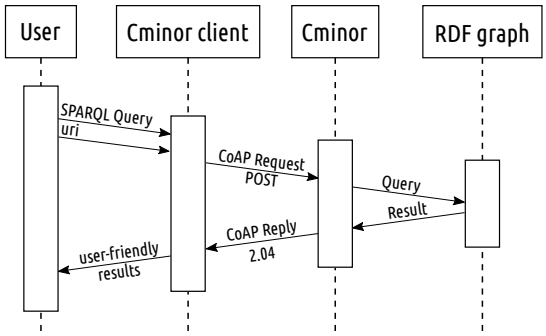

Fig. 3. Sequence diagram for SPARQL queries

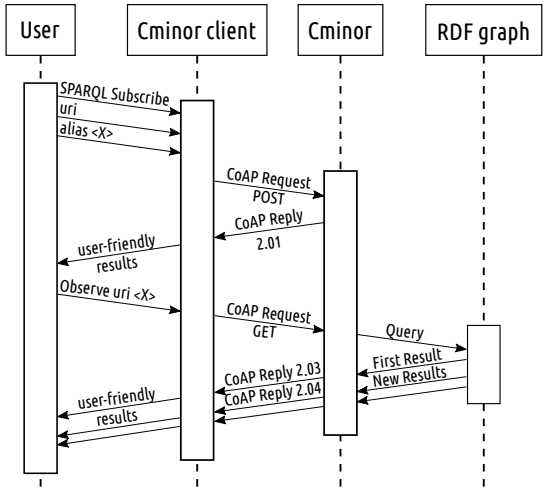

Fig. 4. Sequence diagram for registration of SPARQL subscription and subsequent observation

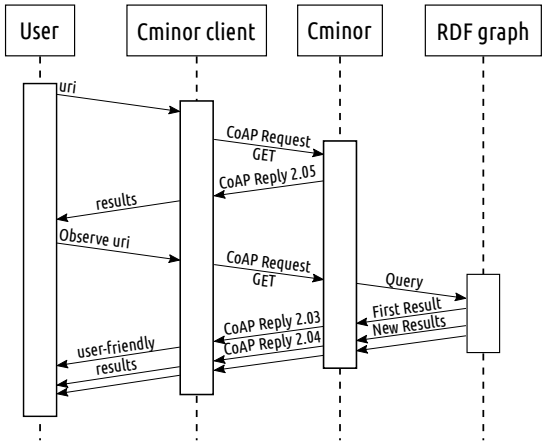

Fig. 5. Sequence diagram for discovery of SPARQL subscriptions and subsequent observation

the broker. The following tests were executed on two C Minor instances: the former, running with an RDFlib store and the latter with a non-persistent instance of Fuseki. As in [31] we made two assumptions:

1) The size of the knowledge base is limited: the KB hosts only the current context. The semantic broker, in fact, should host only the current representation

TABLE II. RESOURCES OF THE C MINOR SERVER

URI	Verb	Payload
/query	POST	the plain SPARQL query
/update	POST	the plain SPARQL update
/subscription	POST	JSON (keys query and alias)
/subscription	DELETE	JSON (key alias)
/<SUB>	GET	-
/.well-known/core	GET	-
/stats	GET	-
/stats	DELETE	-

of the involved resources, which are a direct consequence of the producers involved in the application. In the envisioned IoMusT scenarios, the amount of producers is small if compared to the expected number of consumers.

2) The KB hosts only assertional data and not also terminological data (i.e., the ontology is not stored in the context broker but used by client-side libraries to represent information). Again, this is a technical decision that allows to limit the size of the knowledge base.

According to Gray [54], the evaluation of performance should be relevant to the analyzed use case, simple to understand, portable, and scaleable to assess the performance of small as well as large systems. Keeping this in mind, in this Section we evaluate the performance of C Minor through the typical operations of the IoMusT domain (also employed by the use case detailed later on in Section IV-E). The evaluation has been carried out on a Dell Alienware 17 R2 laptop hosting both the semantic context broker and the clients to minimize the impact of the network (that will be detailed in the following subsections).

A. Evaluation of the Update primitive

Fig 6 and Fig 7 show the behaviour of the broker with respect to SPARQL Update requests with both Fuseki and RDFlib. The test consists in performing SPARQL Updates causing the simultaneous insertion of $n \in [1, 25]$ discrete audio features semantically represented according to the Audio Commons Ontology [55]. The scenario envisioned in this test is that of a set of performers with Smart Instruments sharing the collected audio features with the musical things owned by the audience. With RDFlib the time required to perform the update linearly grows with the complexity of the request. It is easily noticeable that Fuseki outperforms RDFlib fulfilling every request in a nearly constant time inferior to 10 ms. Both the charts report the results of the client-side evaluation (i.e., including the CoAP request and response messages), as well as the time employed by the underlying engine to store data. The average time required by the protocol to dispatch request and response is 3.16 ms.

B. Evaluation of the Query primitive

C Minor with RDFlib shows a better behaviour when responding to query requests, as shown by Fig 9 and Fig 8. This time, the test is aimed at retrieved the whole context that is composed by n discrete audio features semantically mapped according to the Audio Commons Ontology, with $n \in [1, 25]$.

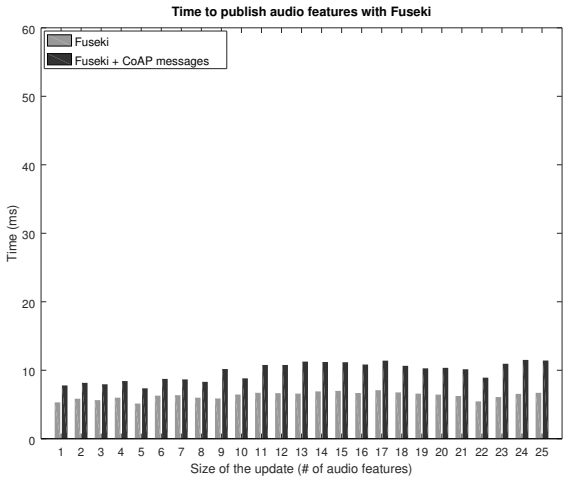

Fig. 6. Time to publish a context composed by n audio features with a SPARQL Update on C Minor + Fuseki ($n \in [1, 25]$).

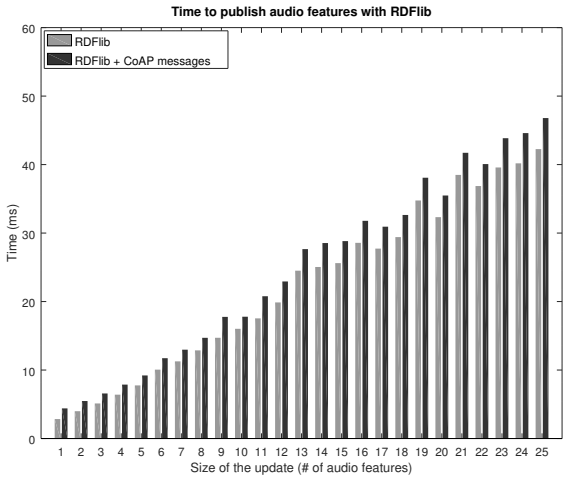

Fig. 7. Time to publish a context composed by n audio features with a SPARQL Update on C Minor + RDFlib ($n \in [1, 25]$).

The higher values in elapsed time running SPARQL queries on C Minor with Fuseki are imputable to the high presence of white spaces and newline characters included by Fuseki when requesting the JSON serialization of results. This ends up with longer messages that require a higher number of UDP segments to be dispatched. The optimization of this step is one of the future works.

C. Evaluation of the Subscription mechanism

Fig. 10 shows the behaviour of C Minor with respect to the subscription mechanism. With this test we aim to quantify the time needed by the server to detect and notify a change to the observers.

The context is composed by an average number of ten audio features. A SPARQL Update request produces the update of a single audio feature. With a subscription to all the audio features running on C Minor, every time the value of a feature changes, the server has to update the state of the subscription

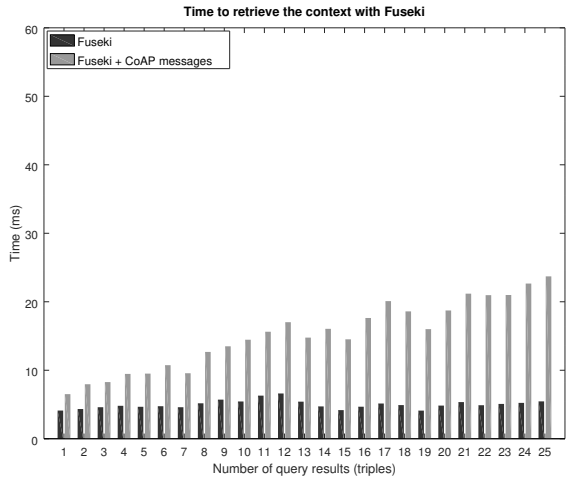

Fig. 8. Time to perform a SPARQL Query on C Minor with Fuseki.

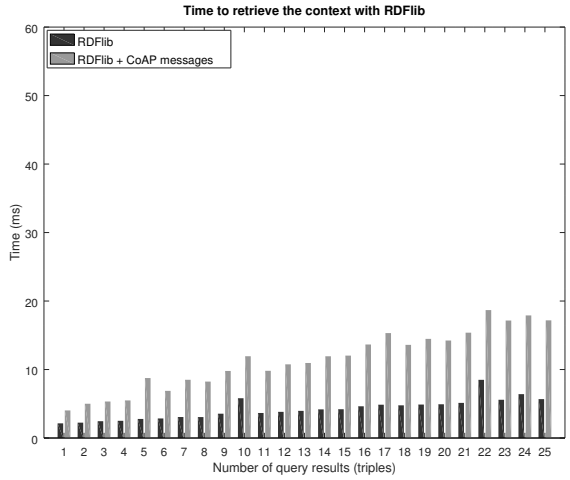

Fig. 9. Time to perform a SPARQL Query on C Minor with RDFlib.

resource monitoring the related subgraph and notify the change to all the observers. In this test the number of observers is n, with $n = 10 \cdot i$ (with $i = 1 \dots 10$).

D. Evaluation of the latency

In this Section we propose an analysis of the latency measured in this test scenario. The evaluation was carried out triggering 100 CoAP requests to C Minor of length n bytes ($n = 20 \cdot i; i = 1, \dots, 25$), and measuring two of the metrics proposed in [56]: the Flow completing time (FCT) and the CoAP Round Trip Time (C-RTT). The first consists of the time interval between the sending of first request and the receiving of the last response. The latter, consists of the average elapsed time between the sending of the original CoAP request and receiving of the CoAP response. This test is intended to measure the latency of the CoAP protocol, so the query, update or subscription mechanisms were not involved in these measurements. Fig. 11 shows that, the time required to complete the flow of requests is nearly constant and is not influenced by the length of the messages. Then, Fig. 12

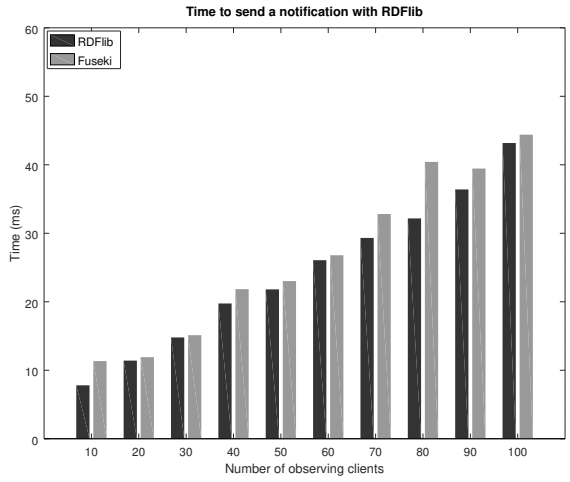

Fig. 10. Time to send a notification to n observers ($n = 10 \cdot i, i = \{1, \dots, 25\}$).

confirms the promising results in terms of latency with average CoAP Round Trip Time inferior to 5 ms.

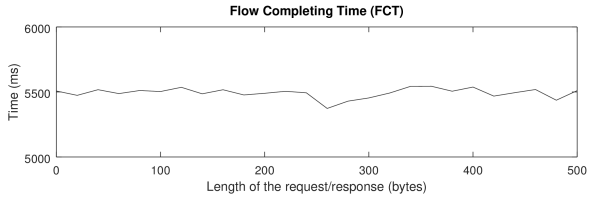

Fig. 11. Flow Completing Time on C Minor

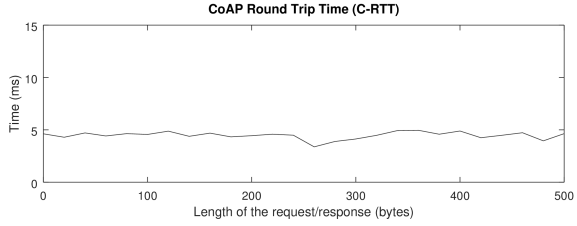

Fig. 12. CoAP Round Trip Time on C Minor

E. Proof of concept

To validate our architecture we implemented a proof-of-concept ecosystem around it, which aimed at simulating the interaction between a smart musical instrument performer and audience members in a TMAP context. The ecosystem, illustrated in Figure 13, comprised the following components:

Smart mandolin. The role of producer was played by the smart mandolin reported in [57]. This instrument consists of a conventional acoustic mandolin smartified with a sensors interface, a contact microphone, a loudspeaker, wireless connectivity, embedded battery, and the Bela board for low-latency audio and sensors processing [58]. Wireless connectivity was achieved by means of the Wi-Fi USB dongle A6100-100PES

by NETGEAR (which supports the IEEE 802.11.ac Wi-Fi standard). Wireless data reception and forwarding were achieved leveraging Open Sound Control (OSC) messages over the User Datagram Protocol.

The audio engine was coded in libpd, a porting of the Pure Data computer music environment into a library for embedded systems [59]. It comprised a variety of ad-hoc sound effects modulating the instrument's string sounds, a library of sound samples to be triggered, as well as mapping strategies to control the sound production from the data gathered from the sensors. In addition, the sound engine comprised a module for the real-time extraction of features from the audio signal captured by the microphone. Specifically, by leveraging the Pure Data object fiddle$\sim$ [60] we extracted the note onset, its pitch and amplitude. From these low-level features we then calculated the note density, the average pitch, and the average amplitudes over 5 seconds.

Musical Thing prototypes. Six prototypes of Musical Things played the role of consumer. They were created in order to simulate an IoMusT scenario involving audience participation (i.e., where audience members use the prototypes to generate musical sounds). Such prototypes were composed by the Bela board, a NETGEAR A6100-100PES Wi-Fi USB dongle, a loudspeaker, and a powerbank. Consumers were given the role of accompaniment of the melody played by the smart mandolin. The sound engine of three prototypes was configured to produce sequences of synthesized notes. Such sequences consisted of random patterns of sounds created with a basic triangular wave generator, whose density was randomized in the range of [1, 200] notes per second. The parameters of each generated note were randomized as follows: the frequency ranged among the frequencies of either the scales G major, D major, or E minor, depending on the prototype; the duration ranged between 10 and 150 ms; the amplitude ranged between 0.01 and 1. The sound engine of the remaining three prototypes was configured to generate one of the following chords: G major, D major, or E minor. These were rendered by means of FM synthesis techniques.

Thanks to a python script communicating via OSC messages to the libpd sound engine, each generated notes sequence or chord was played and stopped according to the notifications issued by the CoAP server. Each of the six consumers was assigned to one of the six statuses "scale G major", "scale D major", "scale E minor", "chord G major", "chord D major", "chord E minor". When the status "silence" was issued by the CoAP server then no chord or note sequence was played and only the melody of the smart mandolin could be heard.

Semantic Server. The semantic server runs on an ODROID-XU4 board (manufactured by Hadkernel), enhanced with the Wi-Fi router TP-Link TL-WR902AC (which features the IEEE 802.11ac standard over the 5GHz band). Following the recommendations reported in [61] to optimize the components of a Wi-Fi system for live performance scenarios to reduce latency and increase throughput, the router was configured in access point mode, security was disabled, and only the IEEE 802.11ac standard was supported.

The server analyzed the information extracted and sent by the smart mandolin as well as delivered to the six consumers the results of the performed analysis. The analysis was based on fuzzy logic techniques [62], where the 27 possible combinations resulting from dividing into 3 parts the range of each of the 3 parameters extracted by the smart mandolin (i.e., note density, the average pitch, and the average amplitudes over 5 seconds), were randomly grouped into 6 subsets of 4 triplets and 1 subset of 3 triplets. The triplets belonging to each subset were then associated to one of the 7 possible statuses: "scale G major", "scale D major", "scale E minor", "chord G major", "chord D major", "chord E minor", "silence".

As previously mentioned, the context broker of the semantic server only hosts the current context. This allows one to increase the performance of the application, since removing outdated information allow the engine to timely process a lower amount of data. More in detail, the context is formed by the audio features published by the smart mandolin and the state produced according to an ad-hoc defined fuzzy logic.

V. Discussion and conclusion

In this paper, we presented C Minor, a semantic broker adapting the concept of a SPARQL Event Processing Architecture [15] to a lightweight application protocol commonly adopted in the IoT: CoAP. The choice of CoAP is due to the need of a direct mapping of the existing SPARQL 1.1 protocol [53] (based on HTTP) on the new broker. To the best of our knowledge, this is the first attempt to build a semantic publish/-subscribe broker on top of the CoAP protocol. Subscriptions can be defined using the SPARQL Query language as in [15]. The main difference consists in the way subscriptions are mapped over the resource/observe paradigm implemented by CoAP, that is intrinsically suitable for grouping equivalent subscriptions and subsequently optimize their processing. This is of paramount importance in applications where a crowd of clients is interested in the same kind of modifications of the graph. On the one hand, this is an undeniable advantage of the C Minor architecture, but, in contrast with the aforementioned SEPA, the entire set of bindings of a triggered subscription is sent after an update.

We demonstrated our approach by implementing an IoMusT ecosystem, where CoAP-speaking clients were embedded in prototypes of smart instruments and Musical Things for audience members. The adoption of semantics in this ecosystem paves the way towards new IoMusT scenarios where 1) heterogeneous devices can interoperate by means of the shared knowledge base and 2) several different applications can co-exist sharing the same context to enable personalized musical experiences. The adoption of CoAP, in this sense, mitigates the poor level of performances of Semantic Web protocols taking advantage of a binary encoding of messages and a faster transport level protocol like UDP and granting low latencies of the exchanged messages. In real IoMusT environments, it is reasonable to think at the audience as people whose Musical Things run applications acting as consumers on the shared knowledge base. It is also very likely that these clients are interested in the same information produced by performers, i.e., they run the same subscription. Then, as the size of the audience (i.e., connected musical things) grows to hundreds of units, it is important to optimize the behaviour of the broker to treat equivalent subscriptions as one unique. As previously mentioned, this is already achieved by C Minor:

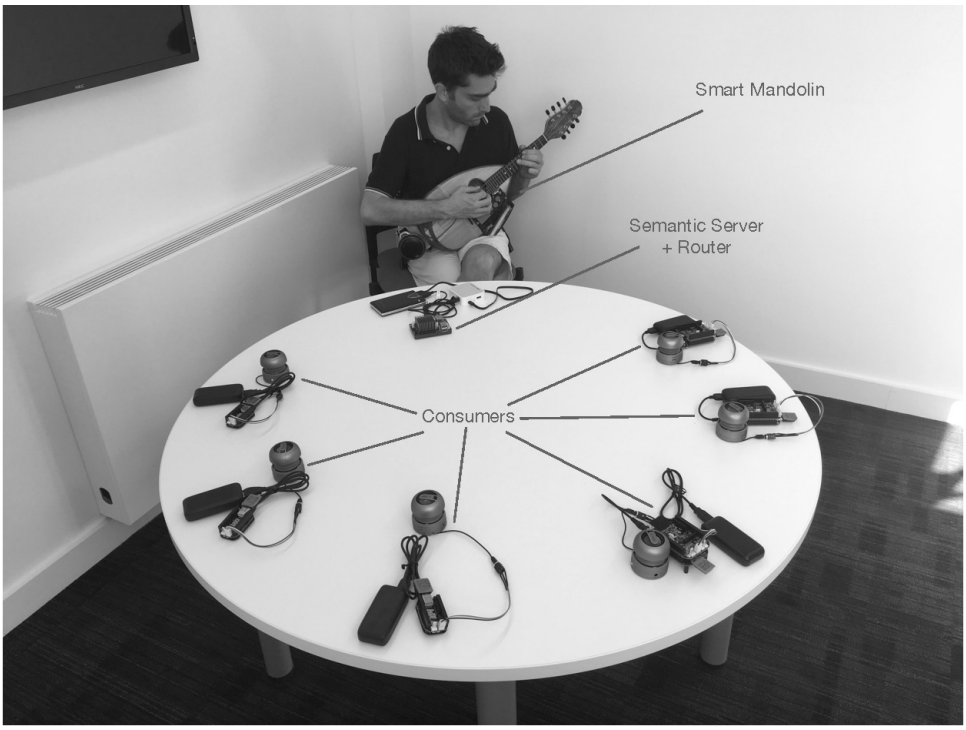

Fig. 13. The developed IoMusT ecosystem with the indications of its hardware components.

equivalent subscriptions are grouped under the same CoAP resource permitting to upsize the scenario.

Regarding future development of the platform, we plan to test multiple solutions to optimize subscription processing. For instance, it is worth mentioning the implementation of the Look-Up tables as suggested in [63]. Look-Up tables would reduce the number of subscriptions to be processed after every update discarding those monitoring a different subgraph. Moreover, starting from the simple tests presented in this paper, we plan to design a benchmark specifically oriented at CoAP-based semantic architectures, in order to achieve a complete evaluation of the performance of the platform varying a higher set of parameters.

ACKNOWLEDGMENT

Luca Turchet acknowledges support from the EU H2020 Marie Curie Individual fellowship (749561), George Fazekas from the EU H2020 Audio Commons grant (688382).

REFERENCES

[1] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–105, 1991.

[2] Kevin Ashton. That 'internet of things' thing. *RFiD Journal*, 22(7), 2011.

[3] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.

[4] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *International symposium on handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[5] Floriano Scioscia and Michele Ruta. Building a semantic web of things: issues and perspectives in information compression. In *IEEE International Conference on Semantic Computing*, pages 589–594. IEEE, 2009.

[6] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[7] Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification, 1998.

[8] Dan Brickley, Ramanathan V Guha, and Brian McBride. Rdf schema 1.1. *W3C recommendation*, 25:2004–2014, 2014.

[9] Deborah L. McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview, 2004.

[10] Andy Seaborne, Geetha Manjunath, Chris Bizer, John Breslin, Souripriya Das, Ian Davis, Steve Harris, Kingsley Idehen, Olivier Corby, Kjetil Kjernsmo, et al. Sparql/update: A language for updating rdf graphs. *W3c member submission*, 15, 2008.

[11] Eric Prud, Andy Seaborne, et al. Sparql query language for rdf. 2006.

[12] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *Systems Engineering Symposium (ISSE), 2017 IEEE International*, pages 1–7. IEEE, 2017.

[13] Dominique Guinard and Vlad Trifa. *Building the web of things: with examples in node. js and raspberry pi*. Manning Publications Co., 2016.

[14] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM computing surveys*, 35(2):114–131, 2003.

[15] Luca Roffia, Paolo Azzoni, Cristiano Aguzzi, Fabio Viola, Francesco Antoniazzi, and Tullio Salmon Cinotti. Dynamic Linked Data: A SPARQL Event Processing Architecture. *Future Internet*, 10(4):36, 2018.

[16] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). Technical report, 2014.

[17] Carsten Bormann, Mehmet Ersue, and Ari Keranen. Terminology for constrained-node networks. Technical report, 2014.

[18] Fabio Viola, Alfredo D'Elia, Luca Roffia, and Tullio Salmon Cinotti. A modular lightweight implementation of the smart-m3 semantic information broker. In *Proceedings of the 18th Conference of Open Innovations Association FRUCT*, pages 370–377. FRUCT Oy, 2016.

[19] Alfredo D'Elia, Fabio Viola, Luca Roffia, Paolo Azzoni, and Tullio Salmon Cinotti. Enabling interoperability in the internet of things: A osgi semantic information broker implementation. *International Journal*

on Semantic Web and Information Systems (IJSWIS), 13(1):147–167, 2017.

[20] Alfredo D'Elia, Fabio Viola, Federico Montori, Marco Di Felice, Luca Bedogni, Luciano Bononi, Alberto Borghetti, Paolo Azzoni, Paolo Bellavista, Daniele Tarchi, et al. Impact of interdisciplinary research on planning, running, and managing electromobility as a smart grid extension. *IEEE Access*, 3:2281–2305, 2015.

[21] Luca Bedogni, Luciano Bononi, Alberto Borghetti, Riccardo Bottura, Alfredo D'Elia, Marco Di Felice, Federico Montori, Fabio Napolitano, Carlo Alberto Nucci, Tullio Salmon Cinotti, and Fabio Viola. An integrated traffic and power grid simulator enabling the assessment of e-mobility impact on the grid: a tool for the implementation of the smart grid/city concept. *Technical Sciences*, 1(1):73–89, 2016.

[22] Luca Turchet, Carlo Fischione, and Mathieu Barthet. Towards the Internet of Musical Things. In *Proceedings of the Sound and Music Computing Conference*, pages 13–20, 2017.

[23] Luca Turchet, Andrew McPherson, and Carlo Fischione. Smart Instruments: Towards an Ecosystem of Interoperable Devices Connecting Performers and Audiences. In *Proceedings of the Sound and Music Computing Conference*, pages 498–505, 2016.

[24] Pratikkumar Desai, Amit Sheth, and Pramod Anantharam. Semantic gateway as a service architecture for iot interoperability. In *IEEE International Conference on Mobile Services*, pages 313–319. IEEE, 2015.

[25] Martin Murth and Eva Kühn. Knowledge-based coordination with a reliable semantic subscription mechanism. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1374–1380. ACM, 2009.

[26] Adel Alti, Abderrahim Lakehal, Sébastien Laborie, and Philippe Roose. Autonomic semantic-based context-aware platform for mobile applications in pervasive environments. *Future Internet*, 8(4):48, 2016.

[27] Sven Schade, Frank Ostermann, Laura Spinsanti, and Werner Kuhn. Semantic observation integration. *Future Internet*, 4(3):807–829, 2012.

[28] Fabio Viola, Alfredo D'Elia, Dmitry Korzun, Ivan Galov, Alexey Kashevnik, and Sergey Balandin. The M3 architecture for smart spaces: Overview of semantic information broker implementations. In *Proceedings of the 19th Conference of Open Innovations Association (FRUCT)*, pages 264–272. IEEE, 2016.

[29] Francesco Morandi, Luca Roffia, Alfredo D'Elia, Fabio Vergari, and Tullio Salmon Cinotti. Redsib: a smart-m3 semantic information broker implementation. In *Open Innovations Association (FRUCT), 2012 12th Conference of*, pages 1–13. IEEE, 2012.

[30] Ivan V. Galov, Aleksandr A. Lomov, and Dmitry G. Korzun. Design of semantic information broker for localized computing environments in the internet of things. In *Open Innovations Association (FRUCT), 2015 17th Conference of*, pages 36–43. IEEE, 2015.

[31] Victor Charpenay, Sebastian Käbisch, and Harald Kosch. μrdf store: Towards extending the semantic web to embedded devices. In *European Semantic Web Conference*, pages 76–80. Springer, 2017.

[32] Michele Ruta, Floriano Scioscia, Agnese Pinto, Filippo Gramegna, Saverio Ieva, Giuseppe Loseto, and Eugenio Di Sciascio. Coap-based collaborative sensor networks in the semantic web of things. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–18, 2018.

[33] Antonio Garrote Hernández and María N Moreno García. A formal definition of restful semantic web services. In *Proceedings of the First International Workshop on RESTful Design*, pages 39–45. ACM, 2010.

[34] Luca Mainetti, Luigi Manco, Luigi Patrono, Ilaria Sergi, and Roberto Vergallo. Web of topics: An iot-aware model-driven designing approach. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 46–51. IEEE, 2015.

[35] Farzad Khodadadi and Richard O. Sinnott. A semantic-aware framework for service definition and discovery in the internet of things using coap. *Procedia Computer Science*, 113:146–153, 2017.

[36] Fares Kayali, Oliver Hödl, Geraldine Fitzpatrick, Peter Purgathofer, Alexander Filipp, Ruth Mateus-Berr, Ulrich Kühn, Thomas Wagensommerer, Johannes Kretz, and Susanne Kirchmayr. Playful technology-mediated audience participation in a live music event. In *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*, pages 437–443. ACM, 2017.

[37] Atau Tanaka. Mobile music making. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 154–156, 2004.

[38] Antonio D de Carvalho Junior, Sang Won Lee, and Georg Essl. Understanding cloud support for the audience participation concert performance of crowd in c[loud]. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 176–181, 2016.

[39] Yongmeng Wu, Leshao Zhang, Nick Bryan-Kinns, and Mathieu Barthet. Open symphony: Creative participation for audiences of live music performances. *IEEE MultiMedia*, 24(1):48–62, 2017.

[40] Oliver Hödl, Geraldine Fitzpatrick, Fares Kayali, and Simon Holland. Design implications for technology-mediated audience participation in live music. In *Proceedings of the Sound and Music Computing Conference*, pages 28–34, 2017.

[41] Nathan Weitzner, Jason Freeman, Stephen Garrett, and Yan-Ling Chen. massMobile-an Audience Participation Framework. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 21–23, 2012.

[42] Gÿorgy Fazekas, Mathieu Barthet, and Mark B. Sandler. *Novel Methods in Facilitating Audience and Performer Interaction Using the Mood Conductor Framework*, volume 8905 of *Lecture Notes in Computer Science*, pages 122–147. Springer-Verlag, 2013.

[43] Ben Bengler and Nick Bryan-Kinns. Designing collaborative musical experiences for broad audiences. In *Proceedings of the ACM Conference on Creativity & Cognition*, pages 234–242. ACM, 2013.

[44] Jason Freeman. Large audience participation, technology, and orchestral performance. In *Proceedings of the International Computer Music Conference*, 2005.

[45] Luca Turchet, Carlo Fischione, Georg Essl, Damián Keller, and Mathieu Barthet. Internet of Musical Things: Vision and Challenges. *IEEE Access*, 2018.

[46] Damián Keller and Victor Lazzarini. Ecologically grounded creative practices in ubiquitous music. *Organised Sound*, 22(1):61–72, 2017.

[47] Luca Turchet. Smart musical instruments: vision, design principles, and future directions. *IEEE Access*, 2018 (in press).

[48] Luca Turchet and Mathieu Barthet. Envisioning Smart Musical Haptic Wearables to Enhance Performers' Creative Communication. In *Proceedings of International Symposium on Computer Music Multidisciplinary Research*, pages 538–549, 2017.

[49] Luca Turchet and Mathieu Barthet. Demo of interactions between a performer playing a Smart Mandolin and audience members using Musical Haptic Wearables. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 82–83, 2018.

[50] Luca Turchet and Mathieu Barthet. Jamming with a smart mandolin and freesound-based accompaniment. In *Proceedings of the 23rd IEEE Conference of Open Innovations Association (FRUCT)*. IEEE, 2018.

[51] Luca Turchet, Fabio Viola, György. Fazekas, and Mathieu Barthet. Towards a semantic architecture for internet of musical things applications. In *Proceedings of the 23rd IEEE Conference of Open Innovations Association (FRUCT)*. IEEE, 2018.

[52] Michele Albano, Luis Lino Ferreira, Luís Miguel Pinho, and Abdel Rahman Alkhawaja. Message-oriented middleware for smart grids. *Computer Standards & Interfaces*, 38:133–143, 2015.

[53] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, and Elias Torres. Sparql 1.1 protocol. *Recommendation, W3C, March*, 2013.

[54] Jim Gray. Database and transaction processing performance handbook., 1993.

[55] Alo Allik, György Fazekas, and Mark B Sandler. An ontology for audio features. In *ISMIR*, pages 73–79, 2016.

[56] Zheng Fei, Fu Baicheng, and Cao Zhen. Coap latency evaluation.

[57] Luca Turchet. Smart Mandolin: autobiographical design, implementation, use cases, and lessons learned. In *Proceedings of Audio Mostly Conference*, 2018.

[58] Andrew McPherson and Victor Zappi. An environment for Submillisecond-Latency audio and sensor processing on BeagleBone black. In *Audio Engineering Society Convention 138*. Audio Engineering Society, 2015.

[59] Peter Brinkmann, Peter Kirn, Richard Lawler, Chris McCormick, Martin Roth, and Hans-Christoph Steiner. Embedding pure data with libpd. In *Proceedings of the Pure Data Convention*, volume 291, 2011.

[60] Miller S Puckette, Miller S Puckette Ucsd, Theodore Apel, et al. Real-

time audio analysis tools for pd and msp. In *Proceedings of the International Computer Music Conference*, 1998.

[61] Thomas Mitchell, Sebastian Madgwick, Simon Rankine, Geoffrey S Hilton, Adrian Freed, and Andrew R Nix. Making the most of wi-fi: Optimisations for robust wireless live music performance. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 251–256, 2014.

[62] Petr Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013.

[63] Luca Roffia, Francesco Morandi, Jussi Kiljander, Alfredo D'Elia, Fabio Vergari, Fabio Viola, Luciano Bononi, and Tullio Salmon Cinotti. A semantic publish-subscribe architecture for the internet of things. *IEEE Internet of Things Journal*, 3(6):1274–1296, 2016.