Publishing LO(D)D: Linked Open (Dynamic) Data for Smart Sensing and Measuring Environments

Fulvio Corno, Faisal Razzak*

Politecnico di Torino, Italy

Abstract

The paper proposes a distributed framework that provides a systematic way to publish environment data which is being updated continuously; such updates might be issued at specific time intervals or bound to some environment-specific event. The framework targets smart environments having networks of devices and sensors which are interacting with each other and with their respective environments to gather and generate data and willing to publish this data. This paper addresses the issues of supporting the data publishers to maintain up-to-date and machine understandable representations, separation of views (static or dynamic data) and delivering up-to-date information to data consumers in real time, helping data consumers to keep track of changes triggered from diverse environments and keeping track of evolution of the smart environment. The paper also describes a prototype implementation of the proposed architecture. A preliminary use case implementation over a real energy metering infrastructure is also provided in the paper to prove the feasibility of the architecture.

Keywords:
Smart environment, Linked (Dynamic) Data, Publishing dynamic data, Semantic data, Publishing energy information

1. Introduction

The last decade saw the emergence of economically viable and efficient sensor technology which can be integrated with appliances across environments, enabling them to sense and measure features around them, i.e., proximity, temperature, luminosity, pressure, electricity, gas, water, etc. This enabled system designers to construct smart sensing and measuring environments [1] and gave rise to computing models in which networks of devices or sensors may interact with each other and with their environments on regular or sporadic moments to reach some predefined goals. These goals may be managing the comfort of residents, providing feedback to the residents over their daily routines, suggesting possible alternatives to the resident's routine, managing efficiently different operations across the environment, etc.

The potential of building diverse applications over the real-time data generated by devices or sensors inside smart environments is huge. For example, in smart homes, device activation can provide the ability to monitor the current state of the system in real time and at the same time allow estimating the energy usage of the environment. In smart metering systems, the real time measurement of electricity in an environment can provide the consumers with a graphical feedback enabling them to follow better and efficient consumption patterns or with computing models that provide suggestions for efficient consumption. Alternatively, it may provide the energy utility the information to make better energy consumption forecasts by taking into account the consumers' needs. The fundamental property of such systems is the potential of supporting various applications over the same set or subset of data generated by the environment.

This potential can be achieved, if the data gathered and generated by networks of devices or sensors follows an open and standard encoding structure for representation, i.e., the data should be machine understandable and

*Corresponding author.
Email addresses: fulvio.corno@polito.it (Fulvio Corno), faisal.razzak@polito.it (Faisal Razzak)

processable. Usually the structure of the environment which models the controllable and uncontrollable elements of the environment (e.g., house plants, walls, floors, rooms etc) does not evolve over time. In contrast, the data gathered and generated by the installed devices or sensors does evolve over time depicting a new overall picture of the environment at each update or change.

Many researchers have used semantic web tools (ontologies) to describe smart environments [2, 3]. An ontology acts as a knowledge base which models the organization and semantics of elements in an environment. While existing semantic tools and reasoning engines deal with time invariant ontological knowledge, supporting rapidly changing information become critical in last couple of years.

This paper proposes a framework providing smart sensing and measuring environments the ability to expose semantically annotated sensor's or device's data being continuously updated; such updates might be issued at specific time intervals or be bound to some environment-specific event. The framework is based upon the publisher-subscriber pattern [4] and is designed to be integrated with an environment, wishing to expose machine understandable data over a unique interface for supporting the development of applications with diverse goals.

The paper is divided into six sections. Section 2 defines the addressed problem and outlines general characteristics of the environment where the framework can be employed. Section 3 explains the proposed framework and outlines an architecture for its implementation. To demonstrate the ability of the proposed framework, Section 4 defines a realistic example, using real-world data. Section 5 compares our technique with some existing techniques proposed in the literature and section 6 concludes the paper and provides future directions.

2. Problem Definition

To better understand the problem and its applicability to smart environments, let us consider a future industrial plant, equipped with a state-of-the-art Energy Management System (EMS). The plant has several energy meters measuring the consumption of electricity, gas and water. The plant has its own electricity generation facility fulfilling its own electricity requirements. The plant consumes the needed electricity and may sell the remaining electricity to the National Energy Grid. The scenario contains some unique characteristics; there are many variables changing over time (representing dynamic data), i.e., electrical generation capacity, local electrical usage, current purchasing price for electrical power, gas consumption, water consumption, etc. The structure of the plant remains constant for a long period (representing static data), i.e., the power generation facility, devices, sensors, location, building structure, etc. If the plant is able to expose its requirements and consumptions in a systematic way, a huge number of potential applications can be build on top. For example, feedback applications can be provided with real time data, which in turn could provide real time analysis for efficient energy usage and future pricing predictions to the user, energy brokers could collect data about the extra electricity available at the plant and the current purchasing price and in turn provide recommendations in real time.

In the above described scenario, some systems (industrial plant, energy broker) are publishing pieces of information being continuously updated and the information is subscribed by some other systems (energy broker, energy provider) to support different applications performing variety of objectives. To differentiate between systems, we describe the former systems as "Publishers" and the latter as "Subscribers". Formally, a "Publisher" is a software application that wishes to expose data (being continuously updated) in an open and standard format having formal semantics. A "Subscriber" represents an application that consumes the data exposed by publishers to achieve some objectives or provide third-party services to users. It can re-publish the data after processing, acting as a publisher, too.

The proposed framework targets scenarios like the one described above, that will enable publishers to publish both the static data and dynamic data of a smart environment to interested subscribers. The framework should be generic in nature and should provide following features:

1. The framework should provide publishers the ability to expose the data being continuously updated; such updates might be issued at specific time intervals (typical update cycles range from a few seconds for device states, to few minutes for energy related information) or might be bound to some environment-specific event.

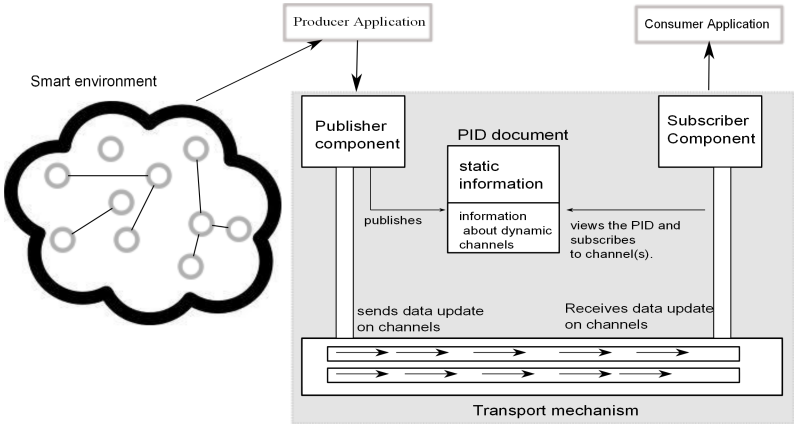

Figure 1: Architecture of the Framework

2. The framework should provide publishers the ability to expose the dynamic aspects of the environment (sensor or device data) separately from the static aspect (the structure of the environment).
3. In order to allow other systems, i.e., automated agents or machines, to understand and process the system on the fly, data should be encoded in an open and standard format with attached semantics.
4. The framework should provide publishers the ability to expose several different streams of data (channels).
5. The framework should provide subscribers the ability to discover different channels exposed by publishers and the structure of the data carried by them.
6. The framework provides subscribers the ability to consume data originating from different publishers.
7. The increase in the number of subscribers should not affect the performance of the publisher.
8. Besides providing the publisher the ability to expose data (corresponding to updates) in an open and standard format, the framework should allow the publisher to define structure of the data carried by channels. For example, the data could be sensor measurement, time of measurement etc.

3. Proposed Framework

Figure 1 depicts the general architecture of the proposed framework. It consists of a publisher, a subscriber components and a transport mechanism. The framework is used by a smart environment willing to publish some data and a consumer application interested in the data of the environment.

The publisher component allows a smart environment (having network of devices or sensors) to publish the data. The environment can use the publisher component to expose the description of the environment which contains the structure of the environment, its different elements (sensor or devices) and their properties, and a description of channels created to carry updated sensor values or device states, in form of a "Publisher Information Document (PID)". The publisher component allows to create channels, which stream updates to interested subscribers in near real time. A channel carries an update in form of a RDF [5] fragment with a set of properties. At each update, the values of the properties in the RDF fragment are updated and the RDF fragment is published on the channel. The set of properties, i.e., measurement value, measurement time, of the RDF fragment and the method to subscribe to the channel is also included in the *PID*. The *PID* is described using a "Publisher" ontology defined latter in Section 3.1.

The subscriber component allows a consumer application to consume published data originating from the publisher. It accesses the PID document to know the structure of the environment and the list of exposed channels. It then allows consumer applications to subscribe to different channels and when updated data arrives at the channel, the subscriber component provides it to the consumer application.

In order to avoid additional resource consumption on the publisher end, as the number of subscribers increases the proposed architecture favors the use of a third-party transport mechanism that keeps the list of subscribers and the

logic of streaming data outside the publisher component. In this regard, we propose the use of cloud based publisher-subscriber services like PubNub[1] and/or Pusher[2].

The underlying theme of the proposed framework is that the producer applications use a publisher component to publish the data using PID and Channels. The consumer application subscribes to the channel(s) and processes the published data, and it could range from being a simple feedback application to an advanced complex event processing system. The tasks of keeping track of subscribers and streaming data to them is handled by the transport mechanism, setting the publisher free of such tasks. Besides being easy to integrate, the publisher is light-weight as the logic of keeping track of subscribers and streaming updates is outside the publisher component.

The details of each component and their working are given below.

3.1. Publisher Component

This component allows a *producer application* to publish structured data. The basic tool to publish data is by defining a PID document, which describes the structure of the environment and the created channels. The structure of the environment can be defined by means of a domain-dependent ontology description, that uses one of the many available ontologies (e.g., for smart homes, it will be structure of the house; for energy management system, it will be installed metering system). The created channels will carry updates. Each update is a RDF fragment having a set of properties, which are dependent on the update being carried by the channel and represent any data which changes over time.

A "Publisher" ontology has been developed to define the formal structure of the *PID document*. It is encoded using the OWL web ontology language[3]. The encoded ontology will provide both publisher and subscriber components a common structure and agreed upon formal semantics. It defines three basic concepts, i.e., Publisher, Static Content, Channel. The structure of the environment is defined using a domain-dependent ontology and pointed by the *Static Content* concept (using hasResource). A channel is described using the *Channel* class and by defining some properties (e.g., for smart environments, the properties may be the name and state of appliances; for energy management systems, the properties could be the power, gas or water consumption along with their measurement units; for sensor networks, the properties may be the location, type, current measured value and unit of measurement of sensors). The information about how to subscribe to a channel depends upon the transport mechanism being used. In case of PubNub, it is a subscription key (added as additional property of the *Channel* class). The *Publisher* class points to both the structure of the environment and all defined channels.

It should be noted that the PID contains only the information about channels, the actual channels are separate transport medium or connections carrying updates, as RDF triples. When an update occurs the publisher component takes the instance of the *Channel* class, fills it with the latest values and send it over a particular channel.

3.2. Subscriber Component

The subscriber component allows a consumer application to access and consume the data published by the publisher. It has the ability to query the *PID document*, extract the defined structure of the environment along with the number of exposed channels. Afterwards, it can subscribe to relevant channels using the subscription key provided for the relevant channel in *PID*. Whenever the publisher component publishes the data over the channel, the subscriber receives it.

4. Use Case: Energy Management Domain

To evaluate the feasibility of the proposed framework, the framework has been implemented as a Java library that includes a publisher and a subscriber component. The publisher component allows to publish the PID document and to create one or more channels. The PID is structured corresponding to the *publisher ontology* and it is published as a web page using the embedded Jetty web server[4]. Channels are created by defining the elements of data (as Channel

[1] http://www.pubnub.com
[2] http://pusher.com
[3] http://www.w3.org/TR/owl-features
[4] http://jetty.codehaus.org/jetty

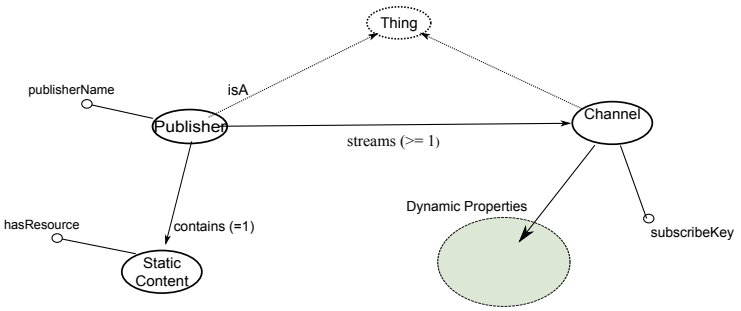

Figure 2: Publisher Ontology

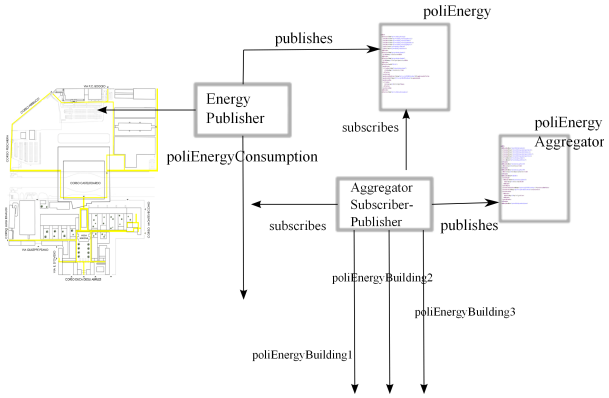

Figure 3: University Metering System Use Case software infrastructure

properties). The elements may be measuring unit, measuring value, time stamp etc, depending on the environment publishing the data. For our current implementation, the publisher component uses PubNub as the transport mechanism, therefore, a *subscription key* is defined for each channel, which will allow the subscriber component to subscribe to a particular channel. All RDF and ontology related operations are handled using the Jena library[5].

To evaluate the feasibility of the proposed framework and its implementation, the main metering infrastructure installed at our university is considered. It is composed of 126 electrical meters. The electrical meters measure the consumption of electrical power (active/reactive) at different locations within the university. Each meter takes the measurements every 15 minutes and stores them in a central database for further analysis. The database contains measurements for over three years and is continuously being updated every 15 minutes. In order to provide real time monitoring of all the meters installed and to analyze measurements from different buildings within the university, we developed a software infrastructure (Figure 3) on top of the installed university metering system, using our proposed framework. The components of the infrastructure are explained below:

1. **Energy Publisher:** Its objective is to provide real time measurement updates from all the electrical meters installed in the university. It initially publishes a *PID document* named *poliEnergy* accessible over HTTP. The metering structure is defined using the DogOnt ontology [3]. It exposes a single data channel, i.e., poliEnergy-Consumption, which carries the updated measurements from all the meters every 15 minutes. A message is sent on the channel for each meter measurement. The defined properties for the *poliEnergyConsumption* channel are meter number, current value, unit and time stamp and the subscription key to subscribe to the channel.
   Table 1 in column *"PID Snippet"* shows a snippet of the *PID* describing the publishing environment, the channel and its elements (properties) describing the information carried by the channel. The column *"Channel snippet"* shows a single message being sent for one electricity meter.

[5]http://incubator.apache.org/jena

PID Snippet

```
<rdf:RDF
    xmlns:RDF="http://www.w3.org/1999/02/22-RDF-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:publisher="http://elite.polito.it/ontologies/Publisher#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/RDF-schema#" >
        <!-- Channel properties --!>

<RDF:Description RDF:about="&Publisher;Unit">
 <rdfs:range RDF:resource="http://purl.oclc.org/NET/muo/ucum/unit/"/>
 <rdfs:domain RDF:resource="&Publisher;Channel"/>
 <RDF:type RDF:resource="&owl;DatatypeProperty"/>
</RDF:Description>

<RDF:Description RDF:about="&Publisher;MeterNumber">
 <rdfs:range RDF:resource="&xsd;integer"/>
 <rdfs:domain RDF:resource="&Publisher;Channel"/>
 <RDF:type RDF:resource="&owl;DatatypeProperty"/>
</RDF:Description>

<RDF:Description RDF:about=
 "&Publisher;hasCurrentValue">
 <rdfs:subPropertyOf RDF:resource="&owl;topDataProperty"/>
 <rdfs:range RDF:resource="&owl;real"/>
 <rdfs:domain RDF:resource="&Publisher;Channel"/>
 <RDF:type RDF:resource="&owl;DatatypeProperty"/>
</RDF:Description>

<!-- Channel Name and Static properties --!>

<RDF:Description RDF:about=
 "&Publisher;poliEnergy_communication">
 <publisher:Location RDF:datatype="&xsd;string">Torino, Italia</publisher:Location>
<publisher:subscribekey>sub-xxxxxx-4290-yyyy-a138-4d46dEEEEE</publisher:subscribekey>
<publisher:channelName>poliEnergy_communication</publisher:channelName>
 <RDF:type RDF:resource="&Publisher;Channel"/>
</RDF:Description>
```

Channel Snippet

```
<rdf:RDF
xmlns:RDF="http://www.w3.org/1999/02/22-RDF-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:publisher="http://elite.polito.it/ontologies/Publisher#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/RDF-schema#" >

<rdf:Description RDF:about="&publisher;poliEnergy_communication">
<publisher:MeterNumber RDF:datatype="&xsd;int">
231
</publisher:MeterNumber>
<publisher:Unit RDF:datatype="&xsd;string">
http://purl.oclc.org/NET/muo/ucum/unit/power-level/bel-kilowatt
</publisher:Unit>
<publisher:hasTimeStamp RDF:datatype="&xsd;dateTime">
2012-02-02T13:06:41.056Z
</publisher:hasTimeStamp>
<publisher:hasCurrentValue RDF:datatype="&xsd;double">
0.3
</publisher:hasCurrentValue>
</RDF:Description>
</rdf:RDF>
```

Table 1: Energy Publisher Information

2. **Aggregator Subscriber-Publisher:** Its objective is to combine the measurements of all the meters coming from the *Energy Publisher* and to provide aggregated measurements for three different buildings within the university. It subscribes to the "poliEnergyConsumption" channel. Every 15 minutes, it is informed about the updated measurements of all the electrical meters, i.e., 126 electrical meters. It aggregates the measurements from electrical meters corresponding to three main buildings and publishes the aggregated electrical measurements using three separate channels, i.e., poliEnergyBuilding1, poliEnergyBuilding2 and poliEnergyBuilding3. The information carried by these channels includes current aggregated measurement and the time stamp.
3. **Desktop Monitoring (Subscriber):** It provides the ability to monitor the aggregated measurements of three buildings coming from the *Aggregator Subscriber-Publisher* in real time. It retrieves the PID of the *Aggregator Subscriber-Publisher* and then subscribes to all three channels. A separate real time graph is shown for each building. Figure 4 shows a snapshot of the application.

5. Related Works

A. Passant et al. [6] proposed sparqlPuSH, a proactive notification system of data updates in RDF stores using PubSubHubbub. The publisher consists of a RDF triple store and subscribers can register different queries on the publisher. Whenever there is an update in the triple store, all the registered queries are evaluated and subscribers are informed about the new or updated results. The list of subscribers is maintained at the publisher end. Our technique is clearly distinct as it advocates a light-weight publisher providing the ability to expose the data being continuously updated, no specific processing is performed at publisher end and most importantly, the list of subscribers is not kept and managed by the publisher but by a third-party cloud based service. In sparqlPuSH [6], as the number of registered queries becomes large, for each triple store update, the time to evaluate all queries and send notification using feeds will become large. In our approach the usage and processing of the data is left at the subscriber end. Although it increases network traffic.

A platform (Sense2Web) to publish Linked Sensor Data for the sensor network community is presented in [7]. It enables users to publish their sensor description data as RDF triples, associate any other existing RDF sensor description data, link to the existing resources on publicly available linked data repositories and make it available to

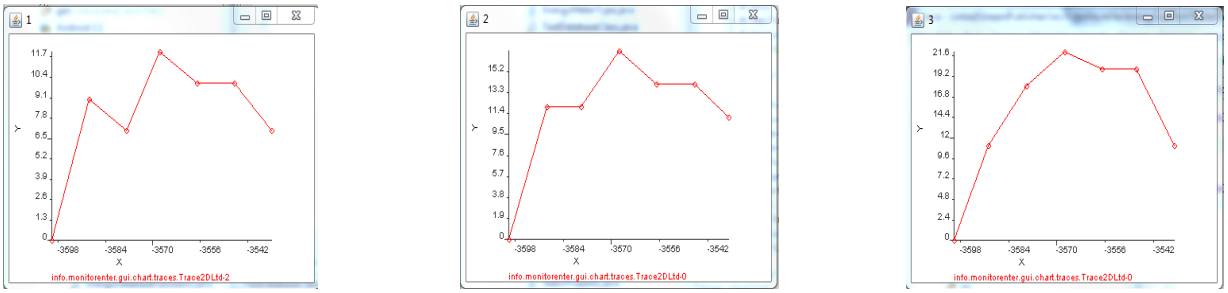

Figure 4: A snapshot of the Desktop Monitoring Application

consumers using SPARQL endpoint [8]. The main focus of Sense2Web is on defining an approach for enriching the sensor description data. Our technique is not an alternate to [7] but is a complement as it focuses more on publishing the updated data, i.e., sensor data. Instead of using the SPARQL endpoint to expose sensor data, which would need to be continuously queried by the consumer to get updates. Our approach will keep subscribers up-to-date.

SEIPF [9] is a semantic energy information publishing framework, which provides the ability to query energy consumption information from residential gateways in a machine understandable format, to achieve consumption coordination and intelligent negotiation. It is based on the client-server model and if queried by the client, gives the consumption of the house based on an Energy Profile ontology. The framework proposed in this paper incorporates the lesson learned from the development of SEIPF. It allows us to move from a client-server model to a publisher-subscriber pattern and allows separation of static information about the environment (using PID document) and dynamic updates (using channels). Moreover, the framework proposed allows easy integration with a smart environment, having characteristics defined in Section 2.

RDF Streaming Engines (EP-SPARQL, SPARQLStream, C-SPARQL, Unified processing of Linked Streams and Linked Data) [10, 11, 12, 13] focus mainly on handling the RDF streaming data from a source. To handle the streaming data, they provide different querying protocols (based on SPARQL) to handle data changes over time. All these techniques mostly focus on the subscriber end. Our proposed technique focuses more on the publisher end, providing a light-weight integration with any source to describe the source and stream its RDF data.

Pachube[6] acts as a central repository where sensors distributed across several locations can upload their data. Data can be uploaded using continuous or non-continuous feeds. A client can download the uploaded data (given access) and use it to achieve its goals. Pachube provides APIs to upload and download data. Our paper also proposes the use of standard APIs to publish and subscribe data, however there is no central repository which stores the data. The transport layer just acts as a transport mechanism. Our model publishes the data in RDF format, so it is easier to build automated subscribers on large scale as the semantics of data are much clearer. The overhead on the client end to continuously query to check latest update and download them in Pachube is overcome by the publisher-subscriber pattern.

6. Conclusion and Future Works

This paper proposed a framework that provides a systematic way to publish sensor rich data from environments. The framework follows general characteristics defined in Section 2. It provides separation of views for the data contained in an environment using the concepts of *PID document* and *Channel*. The *PID document* allows publishers to describe their environments and also provides a systematic way to describe and subscribe to the dynamic channels exposed by the environment for the interested subscribers. The *Channel* carries the data being updated in real time to subscribers. The proposed framework is based on the concept of light-weight publishers, not concerned with keeping track of subscribers. The track of subscribers is kept by a third party cloud based system. The paper also describes the implementation of the framework and a preliminary use case is also detailed in the paper to prove the feasibility of the framework. In future, we plan to evaluate the proposed framework on a large network of nodes subscribed to multiple data channels, processing the data in real time and re-publishing the data.

[6] https://pachube.com/

Acknowledgements

This work is partially supported by Regione Piemonte under project SMILE-O of Polo di Innovazione ICT.

References

[1] M. Weiser, The computer for the 21st century, Scientific American 265 (3) (1991) 94–104.
[2] T. Gu, X. Wang, H. Pung, D. Zhang, An ontology-based context model in intelligent environments, in: Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, 2004, pp. 270–275.
[3] D. Bonino, F. Corno, DogOnt-Ontology Modeling for Intelligent Domotic Evironments, The Semantic Web-ISWC (2008) 790–803.
[4] K. Birman, T. Joseph, Exploiting virtual synchrony in distributed systems, in: Proceedings of the eleventh ACM Symposium on Operating systems principles, ACM, 1987, pp. 123–138.
[5] O. Lassila, R. Swick, Resource Description Framework (RDF) model and syntax, World Wide Web Consortium, http://www. w3. org/TR/WD-rdf-syntax.
[6] A. Passant, P. Mendes, sparqlPuSH: Proactive notification of data updates in RDF stores using PubSubHubbub, in: Scripting for the Semantic Web Workshop (SFSW2010) at ESWC2010, 2010.
[7] P. Barnaghi, M. Presser, K. Moessner, Publishing Linked Sensor Data, in: Proceedings of the 3rd International Workshop on Semantic Sensor Networks, 2010.
[8] E. Prud'Hommeaux, A. Seaborne, SPARQL query language for RDF, W3C working draft 4 (January).
[9] D. Bonino, F. Corno, F. Razzak, Enabling machine understandable exchange of energy consumption information in intelligent domotic environments, Energy and Buildings 43 (6) (2011) 1392 – 1402.
[10] D. Anicic, P. Fodor, S. Rudolph, N. Stojanovic, EP-SPARQL: a unified language for event processing and stream reasoning, in: Proceedings of the 20th international conference on World wide web, ACM, 2011, pp. 635–644.
[11] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, C-SPARQL: SPARQL for continuous querying, in: Proceedings of the 18th international conference on World wide web, ACM, 2009, pp. 1061–1062.
[12] S. Groppe, J. Groppe, D. Kukulenz, V. Linnemann, A SPARQL engine for streaming RDF data, in: Signal-Image Technologies and Internet-Based System, 2007. Third International IEEE Conference on, IEEE, 2007, pp. 167–174.
[13] D. Le-Phuoc, M. Dao-Tran, J. Xavier Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, The Semantic Web–ISWC 2011 (2011) 370–388.