Standardized framework for integrating domain-specific applications into the IoT

Neela Shrestha, Sylvain Kubler, Kary Främling
Aalto University, School of Science and Technology
P.O. Box 15500, FI-00076 Aalto, Finland
Email: firstname.lastname@aalto.fi

***Abstract*—In the so-called "Internet of Things" (IoT), mobile users and objects will be able to dynamically discover and impromptu interact with heterogeneous computing, physical resources, as well as virtual data. It is a fact today that more standardized formats, protocols and interfaces must be built in the IoT to provide more interoperable systems. However, even if a protocol would become the *de facto* IoT standard for data exchange, this would not solve all issues related to system and information integration into the IoT. Indeed, a key challenge is to allow integrating all types of existing domain- or vendor-specific application with such a standard. In this regard, this paper develops a generic framework for such an integration that consists in using the basic communication interfaces supported by the domain-specific application (e.g., basic read and write operations) to extend them into more advanced interfaces provided by IoT standards (e.g., to support subscription mechanisms). In this paper, the set of standardized interfaces defined in a recent IoT standard proposal, referred to as Quantum Lifecycle Management (QLM) messaging standards, are considered for developing the integration framework. A case study using a building-specific application, namely *openHAB*®, is then presented to validate the framework practicability.**

***Index Terms*—Internet of things; Interoperability; System integration; Quantum Lifecycle Management; Intelligent product**

I. Introduction

INTERNET is an evolving entity that started as Internet of Computers, later expanded to Internet of People, and that is now becoming the Internet of Things (IoT) [1]. The IoT envisions a world of heterogeneous objects uniquely identifiable and accessible through the Internet – all forming a dynamic global network infrastructure with self configuring capabilities based on standard and interoperable communication protocols [2]. IoT presents a tremendous opportunity for a multitude of users to be connected to anything, whenever needed, wherever needed [3], [4]. Such opportunities, nonetheless, require the mastery of protocols and standards to make systems interoperable. This is of utmost importance today because numerous competing products and platforms, as well as open-source and proprietary automation applications co-exist together in the IoT [5], [6]. Fig. 1 illustrates various significant IoT domains in which smart and intelligent products are used for various purposes (healthcare assistance, energy saving, building control, maintenance support, facility management…) [4]. The different applications used in each domain as well as components supporting each of these applications are often barely compatible, which is a major hurdle to meet the main IoT requirements [7]. In lack of standardized approaches and protocols, it is likely that a proliferation of architectures, identification schemes and protocols will develop side by side, each one dedicated to a particular or separate use, thus leading to the fragmentation of the IoT [8]. However, rather than fragmenting it, standardized frameworks for integrating distinct domain and sub-domain applications into the IoT should be proposed [9], [10]. These frameworks shall enable information to be conveyed inside and between distinct domains (as illustrated in the "virtual IoT world" in Fig. 1) by taking advantage of the latest generations of standardized IoT interfaces.

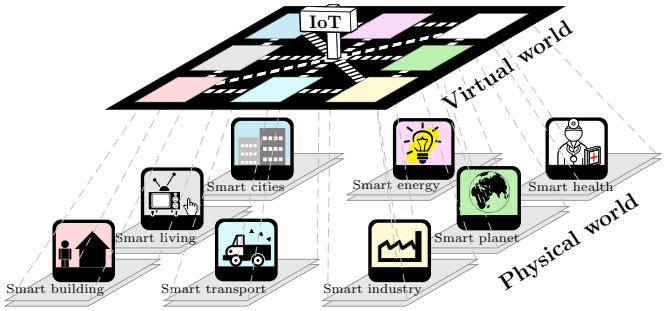

Fig. 1. Significant domains to be integrated into the so-called IoT

This paper investigates and develops such a standardized framework considering a recent IoT standard proposal named Quantum Lifecycle Management (QLM) messaging standards [11]. These standards aim to provide generic and standardized application-level interfaces to enable any kinds of intelligent products to exchange IoT information in ad hoc, loosely coupled ways. Section II provides a high-level description of these standards and the main interfaces aimed to be used in our framework. Section III presents our framework proposal that consists in using the basic communication interfaces supported by the domain-specific application (e.g., basic read and write operations) to extend them into more advanced interfaces provided by QLM (e.g., to support information subscription mechanisms). The originality of this proposal lies in the fact that this framework could be used with any further IoT standard (i.e., other than QLM) that provides similar standardized interfaces. Section IV provides a concrete case study in which we validate the feasibility of using our standardized framework considering a building-specific application named *openHAB* ("smart building" domain, see Fig. 1).

II. QLM MESSAGING

QLM standards emerged out of the PROMISE EU FP6 project, in which real-life industrial applications required the collection and management of product instance-level information for many domains involving heavy and personal vehicles, household equipment, phone switches, *etc.* [12]. Information such as sensor readings, alarms, assembly, disassembly, shipping event, and other information related to the entire product lifecycle needed to be exchanged between several organizations. QLM messaging specifications consist of two standards proposals [11]: the *QLM Messaging Interface* (QLM-MI) that defines what types of interactions between "things" are possible and the *QLM Data Format* (QLM-DF) that defines the structure of the information included in QLM messages. Whereas the Internet uses the HTTP protocol for transmitting HTML-coded information mainly intended for human users, QLM is used for conveying lifecycle-related information mainly intended for automated processing by information systems. In the same way that HTTP can be used for transporting payloads in formats other than HTML, QLM can be used for transporting payloads in nearly any format. XML might currently be the most common text-based payload format but others as JSON and CSV can also be used. The accompanying standard QLM-DF partly fulfills the same role in the IoT as HTML does for the Internet, meaning that QLM-DF is a generic content description model for things in the IoT.

A. QLM Data Format (QLM-DF)

QLM-DF is defined as a simple ontology, specified using XML Schema, that is generic enough for representing "any" object and information that is needed for information exchange in the IoT. It is intentionally defined in a similar manner as data structures in object-oriented programming. QLM-DF is structured as a hierarchy with an "Objects" element as its top element. The "Objects" element can contain any number of "Object" sub-elements, which can have any number of properties, referred to as InfoItems. The resulting Object tree can contain any number of levels. Every Object has a compulsory sub-element called "id" that identifies the Object. The "id" should preferably be globally unique or at least unique for the specific application, domain, or network.

B. QLM messaging interface (QLM-MI)

A defining characteristic of QLM-MI is that QLM nodes may act both as "servers" and as "clients", and therefore communicate directly with each other or with back-end servers in a peer-to-peer manner. Typical examples of exchanged data are sensor readings, lifecycle events, requests for historical data, notifications, *etc.* One of the fundamental properties of QLM-MI is that QLM messages are protocol agnostic so they can be exchanged using HTTP, SOAP, SMTP or similar protocols. In this paper, only interfaces required for the development of our integration framework are presented (an exhaustive list of QLM interfaces is provided in [11]). In this regard, three operations are defined in QLM-MI:

1) *Write:* used to send information updates to QLM nodes;

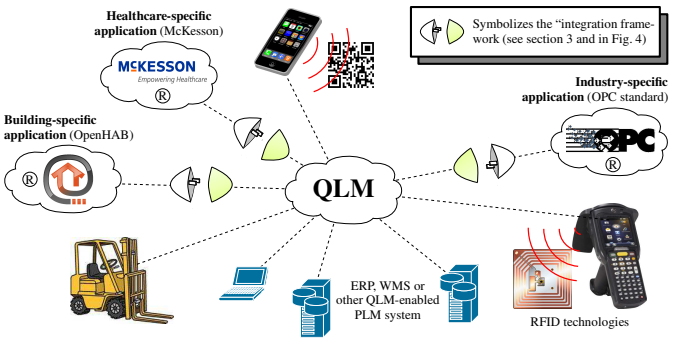

Fig. 2. QLM cloud and standardized framework

2) *Read:* used for **immediate retrieval** of information and for placing **subscriptions** for deferred retrieval of information from a node;
3) *Cancel:* used to cancel a subscription.

The subscription mechanism is a cornerstone of that standard. Two types of subscriptions can be performed:

2a *subscription **with** callback address:* the subscribed data is sent to the callback address at the requested interval. Two types of intervals can be defined: *interval-based* or *event-based*;

2b *subscription **without** callback address:* the data is memorized on the subscribed QLM node as long as the subscription is valid. The memorized information can be retrieved (i.e., polled) by issuing a new QLM read query by indicating the ID of the subscription.

III. INTEGRATION FRAMEWORK FOR IOT STANDARDS

The starting point for integrating any domain-specific application into the IoT is to use a standard providing sufficiently high-level abstraction interfaces as foundation of the integration framework. In this regard, QLM messaging standards are used to take advantage of the wide range of properties and interfaces defined in QLM-DF and QLM-MI. Fig. 2 illustrates the QLM cloud where intelligent products can communicate with each other. Two types of situations might occur:

1) products (or backend systems) implement the QLM messaging standards and can therefore exchange information with each other using the conventional QLM interfaces. Such products are represented in Fig. 2 by the smartphone, RFID reader, databases, and the crane;
2) products are located in a domain- or network specific application and are thus unable to exchange information with any other node located in the QLM cloud. Three domain-specific applications are illustrated in Fig. 2, namely from the sector of building (*openHAB*), healthcare (*McKESSON*), and industry (*OPC*), but many more could be cited in many other domains [13] (see Fig. 1).

As previously stated, the main idea of our integration framework is to extend the basic communication interfaces supported by the domain-specific application into more advanced interfaces supported by the IoT standard (i.e., QLM in this study). In a more concrete level, the QLM node at the border between the two "worlds" (such a border is

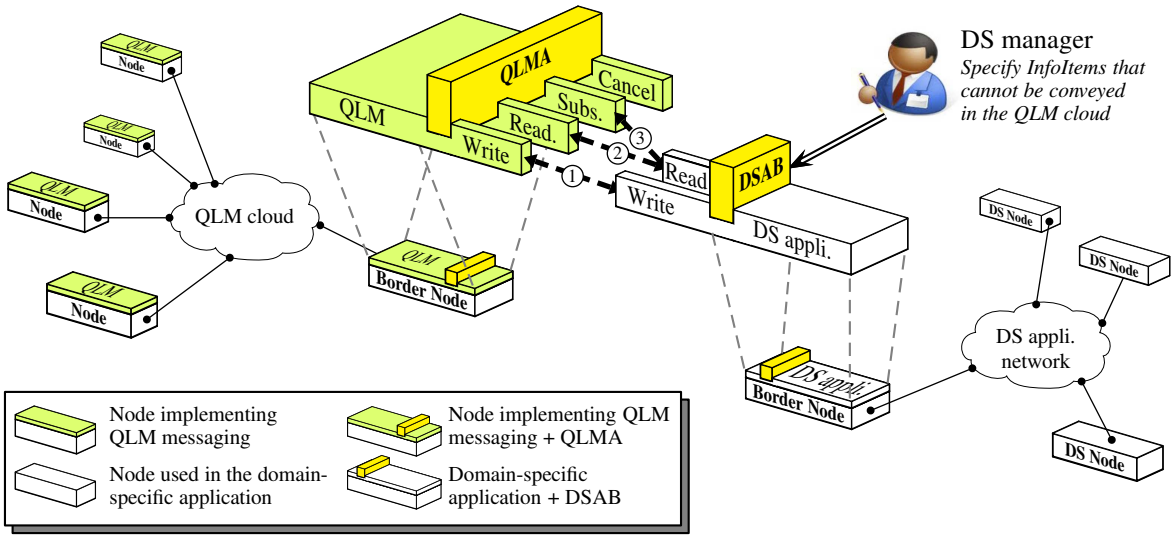

Fig. 3. View on the standardized framework for integrating domain-specific applications into the QLM cloud

represented by the different "sockets" in Fig. 2), which is denoted by "border QLM node" in Fig. 3, is in charge of invoking/soliciting the appropriate interfaces of the DS node (denoted by "DS border node") depending on the user request (read, write, subscription, or cancel). For this to happen, two components are defined in our framework:

- *Domain-Specific Application Binding (DSAB):* in our framework, it is necessary that the domain-specific application provides, at least, two basic operations as illustrated in Fig. 3: "read" and "write" to respectively retrieve and modify a particular information in the domain-specific network;
- *QLM Agent (QLMA):* the agent embedded on the border QLM node is in charge of soliciting the DSAB component according to the requested QLM operations (see items 1, 2, 3, 2a, and 2b in section II-B). For instance, if a QLM node (outside the domain-specific network) wants to subscribe to an information located in the domain-specific network (e.g., with an interval of t sec), QLMA should invoke every t sec the read operation of DSAB (i.e., to poll the InfoItem value at the requested interval). The dashed arrows drawn between QLMA and DSAB in Fig. 3 highlight how QLM operations are linked to the basic operations supported by the specific application.

Developments made in QLMA are independent of the domain-specific application, while developments made in DSAB are not (e.g., commands to retrieve and modify data in *openHAB* certainly differ from the ones defined by *OPC*). Section III-A introduces the principle of "extension" used in our framework to link the "read" and "write" operations between QLMA (i.e., the QLM cloud) and DSAB (the domain-specific network), which correspond to arrows denoted by ① and ② in Fig. 3). Section III-B presents such an extension to support the two QLM subscription mechanisms (corresponds to arrow denoted by ③ in Fig. 3). Finally, section III-C lays down conditions of security to convey particular information from the domain-specific network to the QLM cloud.

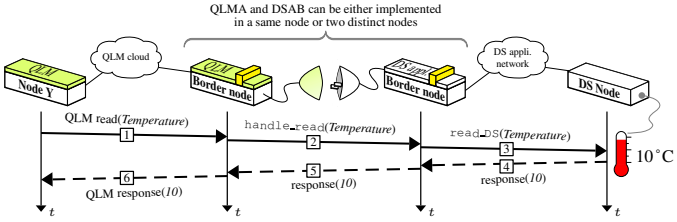

Fig. 4. Sequence diagram considering a QLM read operation

A. Principle of "extension" for Read and Write operations

When a QLM node requests for reading or writing an InfoItem value in the domain-specific application, the corresponding request has to be conveyed to a node that integrates our framework, which is referred to as "border QLM node". This is illustrated through arrow "1" in Fig. 4 considering a "read" query requesting for the value of InfoItem named `Temperature` (the process is similar considering a write query). On receiving the QLM request message, QLMA (i.e., the border QLM node) extracts all the necessary information (InfoItem names, type of operation requested, interval parameter, *etc.*) in order to properly solicit the DSAB component. More concretely, QLMA must start exchanging information with DSAB in other to:

- identify whether the requested InfoItems exist in the domain-specific application and if they are allowed to be conveyed in the QLM cloud (could have read-only or read/write access);
- to invoke the appropriate command(s) (read or write) at the right time on DSAB;
- to return the requested value(s) to the requestor, or an error code if the operation cannot be achieved.

These stages respectively correspond to arrows denoted by "2", "3", "4", "5", and "6" in Fig. 4 and are mainly fulfilled by the function named `handle_read()` (see arrow "2"). This function is detailed in Algorithm 1, where the function input II_n is the name of the requested InfoItem, and the function

Algorithm 1: `handle_read` (II_n)

```
   output: R
1  begin
2      A ← get_list();   // QLMA retrieves from DSAB the list
       of InfoItems allowed to be accessed
3      if II_n ∈ A then                  // If InfoItem II_n is allowed
4          R ← read_DS(II_n);            // QLMA invokes the Read
           operation of DSAB and thus receives a response R
5      else                // If II_n is not allowed to be accessed
6          R ← qlm_error();   // a QLM response with an error
           code is returned to the requestor QLM node
```

Algorithm 2: `handle_sub` (II_n,*inter*,*TTL*)

```
   output: R
1  begin
2      A ← get_list();
3      if II_n ∈ A then
4          if inter > 0 then        // "interval-based" subscription
5              repeat
6                  foreach inter do
7                      R ← read_DS(II_n);
8                      return R; // II_n value is returned every
                       interval of time (noted inter)
9              until TTL expires;
10         else                          // "event-based" subscription
11             repeat
12                 R ← read_DS(II_n);
13                 foreach event do
14                     return R;    // II_n is returned each time
                       it changes (i.e., an event occurs)
15             until TTL expires;
16     else
17         R ← qlm_error();
```

output R is the QLM response returned to the requestor. The function used when receiving a QLM write request, named `handle_write()`, is not presented in this paper but is similar to `handle_read()` except that *i)* the new value to be updated is provided as input parameter of the function, and *ii)* the "read" command (*cf.* row 4 in Algorithm 1) is replaced by the "write" command supported by the application.

B. Principle of "extension" for subscription operations

A QLM node is now able to request for different types of subscriptions to InfoItems located inside the domain-specific application using our framework. Let us now consider a subscription query including the interval parameter, the TTL (period of validity of the subscription), the callback address, and the InfoItem(s) to be subscribed (see arrow denoted by "1" in Fig. 5). When the "border QLM node" receives the subscription query, two tasks are achieved:

- *creating the subscription:* the border QLM node has to create the subscription and to return to the requestor the related subscription ID. This ID is useful, among others, to cancel the subscription before it expires;
- *handling of the subscription:* the border QLM node has to retrieve the subscribed InfoItem value from the domain-specific application every *interval* of time (see arrows denoted by "3", "4", "5"), and then to push it to the subscriber node (i.e., the callback address) .

Both tasks are fulfilled by the function named `handle_sub()` (see arrow "3"). This function is detailed in Algorithm 2, where the function inputs II_n is the name of the requested InfoItem, *inter* is the interval parameter specified by the user (in sec), TTL is the period of validity of the subscription (in sec), and the function output R is the QLM response returned to the subscriber. As previously mentioned, two types of subscriptions can be performed *i)* interval-based or *ii)* event-based. The `handle_sub()` function dissociates both cases through the *If-Else* condition at rows 4 and 10 respectively in Algorithm 2. Then, if it is an interval-based subscription, QLMA invokes the read command in the domain-specific application each interval of time (see rows 5 to 9); if it is an event-based subscription, the process is similar to the *interval-based* subscription except that a short-polling interval (configurable according to the user needs and value change latency, but usually on a millisecond time scale) is used to read the subscribed value in the domain specific-application. It is therefore trivial to identify when the subscribed value changes, denoted *event* in row 13 in Algorithm 2.

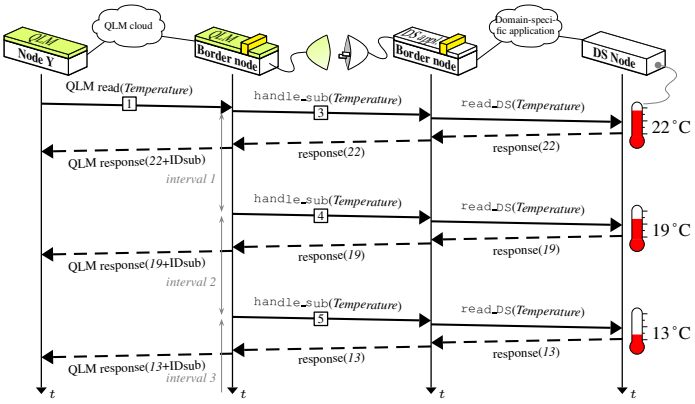

Fig. 5. Sequence diagram considering a QLM subscription operation

Finally, considering a subscription request *without* callback address (whether interval-based or event-based), the principle is the same except that values retrieved by the border QLM node (i.e., QLMA) are memorized by that one. Then, the subscriber node (i.e., QLM node Y in Fig. 5) is able to retrieve historical values by issuing a classical read request by specifying the ID of the subscription and (optionally) two points in time to retrieve particular values.

C. Security policy

An important aspect to take into account is the security and privacy of the information when developing such frameworks. Indeed, a procedure should enable the user, or the manager, of the domain-specific application to permit or deny the exchange of information from his network to the QLM cloud, and *vice-versa*. This is especially true for applications with a high degree of privacy like healthcare applications (e.g., *McKESSON*). Accordingly, a supplementary condition is added to our framework, which is that the domain-specific application must provide the functionality to "bind" one or several InfoItems from the domain-specific network to the

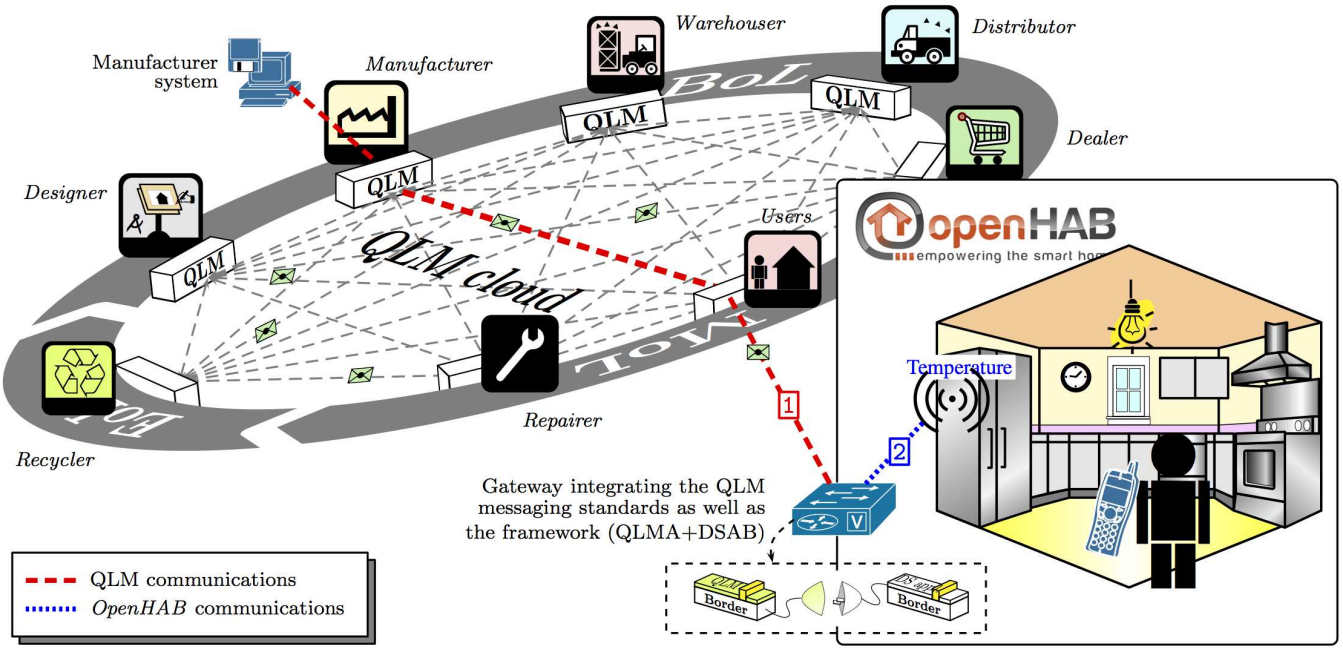

Fig. 6. Home automation platform implementing the standardized framework for integrating *openHAB*® into the QLM cloud

DSAB agent. In other words, if an InfoItem is bound to DSAB, it is allowed to be exchanged with any system located in the QLM cloud, depending on the access rights currently assigned (read-only or read/write access). Such a procedure is highlighted in Fig. 3 with the DS manager.

IV. Home Automation Platform

This section presents a case study to demonstrate the validity of our framework. This case study is defined in the framework of smart building (*cf.* Fig. 1), in which the *openHAB*® platform is used by a resident to control various home appliances in his house (e.g., fridge, TV, ventilation system…). Such a house and *openHAB* environments are depicted in Fig. 6. The user just acquired a smart fridge[1] and bought related services that were proposed during the purchase process. One of these services includes the continuous monitoring of the fridge temperature, detection of potential failures (e.g., the indoor temperature increases abnormally), and proactive decision-making (purchase of spare units, scheduling of repair orders…). This service is carried out by the fridge manufacturer (see Fig. 6), but the installation and setting up are achieved by the Dealer, who subsequently provides the necessary information to the manufacturer (IP address of the resident, *etc.*). In our case study, QLM messaging standards and the framework (QLMA and DSAB) are implemented on the Gateway access point of the resident as illustrated in Fig. 6. This does not impact the existing *openHAB* communications, it only enables the access from the QLM cloud to information consumed or generated by the *openHAB* network.

At this point, QLM and the framework are ready for use but it is still necessary to specify which information from *openHAB* is allowed to be conveyed into the QLM cloud

[1]A real fridge has been fitted with sensors and actuators in our platform.

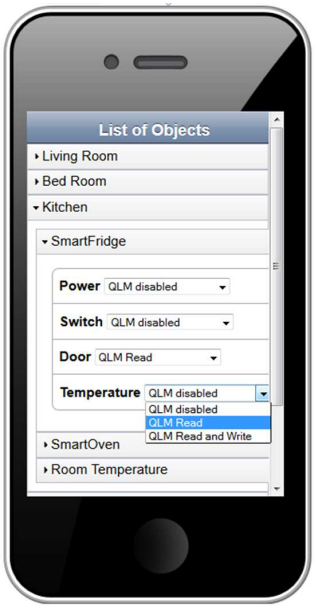

Fig. 7. Application screenshot related to the security procedure

(i.e., the security procedure introduced in section III-C). Fig. 7 provides a screenshot of the smartphone application currently being developed to enable the user (or Dealer during the installation) to select such information. This screenshot shows the rooms of the house (Living room…), the appliances available (or reachable using *openHAB*) in each room (e.g., *Smartfridge* in the kitchen), as well as the different information (referred to as InfoItems in our study) that is accessible on each appliance. In our case study, four InfoItems can be accessed on the smart fridge, as shown in Fig. 7, and that two of those InfoItems (*Temperature*, *Door*) were assigned to read only,

while the others are not allowed to be accessed from the QLM cloud.

Section IV-A provides greater detail about the communications between manufacturer information system and the fridge, and shows how this service could be adapted in "real-time". A short analysis and discussion is then proposed in section IV-B.

A. Continuous monitoring of the fridge

Once the manufacturer information system received all the necessary information from the Dealer, namely the IP address of the resident's gateway access point, it can start exchanging information with this gateway in order to fulfill the service requested by the resident. Since the the manufacturer system is not aware about the InfoItem names, it uses the RESTful QLM "discovery" mechanism (see [11]) for identifying what "Object" and related InfoItems can be accessed from the QLM cloud. Fig. 8 shows how this discovery mechanism can be invoked using the Unix `wget`. The manufacturer system executes `wget_1` (made of the IP address of the resident gateway) that returns the set of "Object"(s) (their respective ID to be accurate) related to the house, i.e. Objects that are reachable using QLM. It can be observed that three Object IDs are returned, which correspond to different rooms (Living Room, Kitchen...). Based on those information, the manufacturer system refines his research using `wget_2` by retrieving the set of sub-elements (Objects or InfoItems) contained in the Object `Kitchen10` (`wget_2` is the concatenation of `wget_1` with this ID). Two sub-"Object"(s) are returned that correspond, in our platform, to the set of appliances reachable with QLM, namely the smart fridge and smart oven (see Fig. 8). Finally, the manufacturer system identified the fridge it was looking for, but still needs to identify what information is reachable in *openHAB* (the system is loocking for the InfoItem that provides the fridge temperature, which is required to successfully provide the requested service). By executing `wget_3` (see Fig. 8), the list of InfoItems that can be accessed on that appliance is returned. In this case, two InfoItems are accessible in "read only" mode[2] that are the ones previously selected during the security procedure (specified by the user or Dealer as shown in Fig. 7), which are the fridge temperature whose name is `Temp_sensor22` and the door status whose name is `Door_sensor1` (see Fig. 8).

Based on these information, the manufacturer system then sends a QLM subscription query to the QLM border node (i.e., the resident's gateway) for subscribing to `Temp_sensor22`. The XML code related to this query is shown in Fig. 9, in which the QLM operation is specified in row 2 (i.e., in the messaging interface QLM-MI). This operation is set to "read" since a subscription is a particular read, the interval parameter is set to 300 (expressed in seconds; i.e. 5 min), and the callback address corresponds to the manufacturer system (see rows 2 and 3 in QLM-MI). In the message body (i.e., QLM-DF), the InfoItem that needs to be subscribed (i.e., `Temp_sensor22`) is specified in row 8.

[2]*Metadata* provides more meaningful information about the InfoItem, such as value type, units and other similar information.

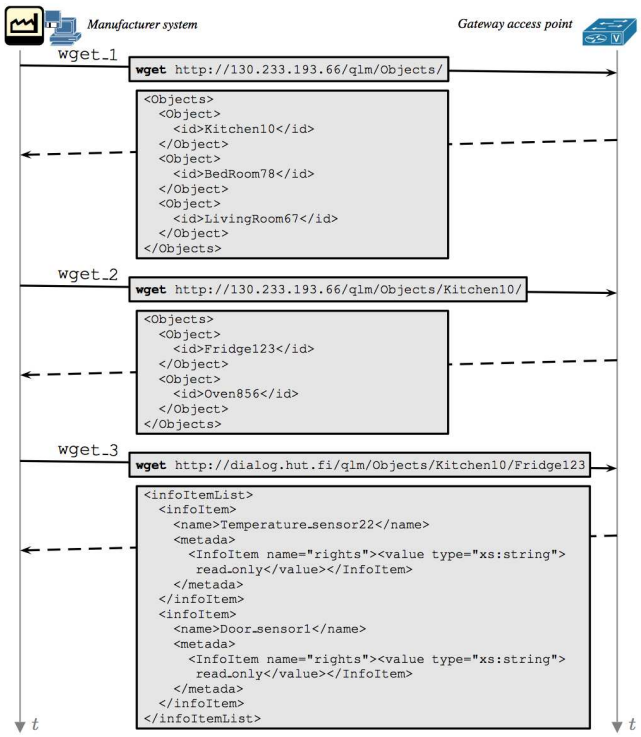

Fig. 8. RESTful QLM "discovery" mechanism used by the manufacturer system

```
1  <qlmEnvelope xmlns="QLMmi.xsd" version="1.0" ttl="-1.0">
2    <read msgformat="QLM_mf.xsd" interval="300" callback=
3      "http://130.233.193.149/Servlet/receiver">          QLM-MI
4      <msg>
5        <Objects xmlns="QLMdf.xsd">
6          <Object type="Refrigerator">
7            <id>Fridge123</id>
8            <InfoItem name="Temp_sensor22">
9            </InfoItem>
10         </Object>
11       </Objects>                                        QLM-DF
12     </msg>
13   </read>
14 </qlmEnvelope>                                         QLM-MI
```

Fig. 9. QLM subscription query sent by the manufacturer to subscribe to InfoItem `Temperature` to the Fridge (located in the *openHAB* network)

Communications resulting from this subscription query are depicted in the form of a sequence diagram in Fig. 10 jointly with a screenshot showing the traffic captured on the resident's gateway. The arrow denoted by "1" corresponds to the query sent by the manufacturer information system (see also number "1" in the screenshot). When receiving this query, function `handle_sub()` is solicited on the gateway to properly retrieve the subscribed InfoItem from *openHAB* (see Fig. 10). Let us note that no traffic is captured by the network protocol analyzer coming from or going to the smart fridge (i.e., in the *OpenHab* network). The reason is that the communication between the gateway and the fridge (sensors to be exact) is based on the *1-wire* protocol and not 802.x standards. Once the value of `Temp_sensor22` is returned by the fridge to the gateway (i.e., *Temp0* in Fig. 10), the gateway has to push the value to the manufacturer (see arrow denoted by "2") through a QLM response by including the ID of the subscription.

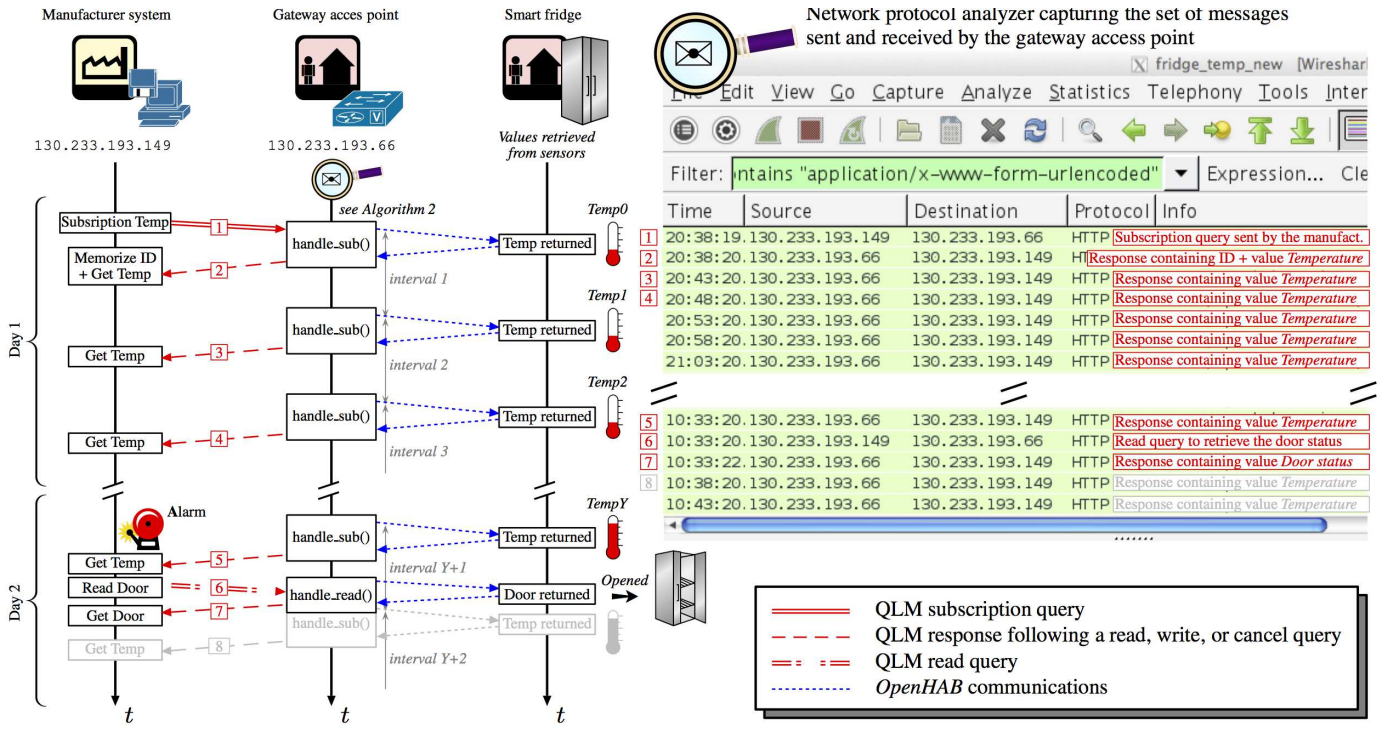

Fig. 10. Sequence diagram related to the home automation platform & Capture of the traffic sent and received by the gateway access point

```
1  <qlmEnvelope xmlns:qlm="QLM_mi.xsd" version="1.0" ttl="
       0.0">
2    <response>
3      <result subscriptionID="SUB_1384939801707">
4        <return returnCode="200">
5          <Objects>
6            <Object>
7              <id>Fridge123</id>
8              <InfoItem name="Temp_sensor22">
9                <value>5</value>
10             </InfoItem>
11           </Object>
12         </Objects>
13       </return>
14     </result>
15   </response>
16 </qlmEnvelope>
```

Fig. 11. QLM response resulting from the subscription performed beforehand by the manufacturer

This response is detailed in Fig. 11 where the subscription ID is provided in row 3, the returned code "200" (row 4) is a default value indicating the success of the operation, and the value of the current temperature is given in row 9 (5 °C). Subsequently, at the requested interval (i.e., every 5 min), the gateway retrieves the temperature (see *Temp1*, *Temp2*... in Fig. 10), which is then transmitted to the manufacturer system using similar QLM response messages (see arrows denoted by "3", "4"...). The traffic captured by the network protocol analyzer shows exactly that every 5 min a new QLM response is generated by the gateway (see the column named "Time" in this screenshot).

During the second day (see "Day 2" in Fig. 10), the fridge door was left open, leading to an increase of the fridge temperature. A wide range of actions can therefore be undertaken by the manufacturer such as the notification of the problem to the user, a further in-depth investigation of the problem, and so on. In our platform, the manufacturer information system carries out further investigation to identify the root causes of the problem, to the extent possible (the manufacturer is not aware about the current context of the fridge like the fact that the door was left open, a heating source has been placed beside the fridge...). The manufacturer system Fig. 10 uses the RESTful QLM "discovery" mechanism for identifying what additional information can be accessed on the fridge (see Fig. 8) and, based on this information, the system sends a QLM read query to retrieve the status of the fridge door (i.e., a read query using InfoItem `Door sensor1`). This action corresponds to arrow denoted by "6" in Fig. 10. This time, the gateway invokes the function `handle_read()` and not `handle_sub()`, which leads to a unique response as shown with arrow denoted by "7" (see also the network protocol analyzer).

B. Discussion and Analysis

As previously mentioned, more complex decision making algorithms could be designed by the manufacturer. Although the information collected in *openHAB* is, at the current stage, only used by the manufacturer, the flexibility of QLM makes it possible to further extend this platform to other purposes/phases of the fridge lifecycle. The collected information could, for instance, be transmitted in immediate or deferred ways to repair and design companies, which could lead to improved maintenance scheduling, product design and manufacturing procedures [14]. This is directly linked to the concept of CL_2M (Closed Loop Lifecycle Management)[3]

[3]www.cl2m.com

[15], [16], [17] whose breakthrough challenge is to enable the information flow to include customers and to enable the seamless transformation of information to knowledge. More generally, more advanced services and interactions among product stakeholders are now possible using QLM, but also thanks to generic frameworks, as the one developed in this paper, that make it possible to integrate any domain-specific application into the IoT. This is an important step to develop ideas for new environment-friendly products and to improve the customer experience.

V. Conclusion

Networks, systems and products of the 21^{st} century transcend the traditional organizational boundaries since every "thing" becomes literally "connected" to the so-called IoT. This vision is getting closer to become real due to the every day increase of concepts and technologies such as sensor hardware/firmware, new communication technologies, semantic models/tools, cloud processing and data storage, and machine learning. Billions of devices are connected to the Internet and it is predicted that there will be $50 - 100$ billions by 2020. Nonetheless, there are still important challenges ahead, and particularly the proposal and adoption of IoT standards that are sufficiently generic and generally applicable for exchanging any kinds of information between any kinds of objects/products. The importance of this standardization work for both academic and commercial purposes should not be under-estimated because the lack of a satisfactory application-level messaging standard is still a great obstacle for realizing the IoT.

Recently, a new IoT standard proposal referred to as "QLM messaging standards" have been developed and proposed to fulfill such requirements, thus providing standardized application-level interfaces. However, even if such a messaging protocol would become the *de facto* standard in the IoT, this would not solve all problems related to system and information integration into the IoT. Indeed, a key challenge today that cannot be addressed by a unique IoT standard is to allow integrating all types of existing domain- or vendor-specific applications into the IoT. Given this observation, this paper develops a generic framework for such an integration, which consists in using the basic communication interfaces supported by the domain-specific application (e.g., basic read and write operations) to extend them into more advanced interfaces provided by IoT standards or, to be more exact, by QLM messaging standards (e.g., to support subscription mechanisms). A case study using a building-specific application, namely *openHAB*®, is used in this paper to validate the framework practicability. This case study shows how such a framework contributes to improve maintenance scheduling, product design, manufacturing procedures, and other activities related to the product lifecycle.

Official QLM messaging specifications are expected to be made public by The Open Group during 2014. However, creating such standards and getting them into widely use can be a long and challenging task, as shown e.g. for the EPC standards [18]. Indeed, the specification of a "good" standard is far from being a simple engineering task. The current QLM messaging specifications are a result of over ten years of research work jointly with many academic and industrial partners. It is nonetheless important to note that the integration framework proposal developed in this paper could be used with any further IoT standard (i.e., other than QLM) that provides similar interfaces.

References

[1] L. Coetzee and J. Eksteen, "The internet of things-promise for the future? an introduction," in *IST-Africa Conference Proceedings*, 2011, pp. 1–9.

[2] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the Internet of Things," *Cluster of European Research Projects on the Internet of Things, European Commision*, 2010.

[3] K. Ashton, "Internet things – mit, embedded technology and the next internet revolution," in *Baltic Conventions, The Commonwealth Conference and Events Centre, London*, 2000.

[4] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[5] T. S. Dillon, H. Zhuge, C. Wu, J. Singh, and E. Chang, "Web-of-things framework for cyber–physical systems," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 9, pp. 905–923, 2011.

[6] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[7] O. Mazhelis, E. Luoma, and H. Warma, "Defining an internet-of-things ecosystem," in *Internet of Things, Smart Spaces, and Next Generation Networking*, 2012.

[8] Q.-B. Sun, J. Liu, S. Li, C.-X. Fan, and J.-J. Sun, "Internet of things: Summarize on concepts, architecture and key technology problem," *Journal of Beijing University of Posts and Telecommunications*, vol. 3, pp. 1–9, 2010.

[9] A. Koubâa and B. Andersson, "A vision of cyber-physical internet," in *Proceedings of the Workshop of Real-Time Networks*, 2009.

[10] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications surveys & Tutorials*, no. 99, pp. 1–41, 2013.

[11] K. Främling and M. Maharjan, "Standardized communication between intelligent products for the IoT," in *11th IFAC Workshop on Intelligent Manufacturing Systems, São Paulo*, 2013, pp. 157–162.

[12] D. Kiritsis, A. Bufardi, and P. Xirouchakis, "Research issues on product lifecycle management and information tracking using smart embedded systems," *Advanced Engineering Informatics*, vol. 17, no. 3, pp. 189–202, 2003.

[13] A. Arsénio, H. Serra, R. Francisco, F. Nabais, J. Andrade, and E. Serrano, "Internet of intelligent things: Bringing artificial intelligence into things and communication networks," in *Inter-cooperative Collective Intelligence: Techniques and Applications*, vol. 495. Springer Berlin Heidelberg, 2014, pp. 1–37.

[14] S. Karjalainen, "Consumer preferences for feedback on household electricity consumption," *Energy and Buildings*, vol. 43, no. 2-3, pp. 458–467, 2011.

[15] D. Kiritsis, "Closed-loop PLM for intelligent products in the era of the internet of things," *Computer-Aided Design*, vol. 43, no. 5, pp. 479–501, 2011.

[16] K. Främling, J. Holmström, J. Loukkola, J. Nyman, and A. Kaustell, "Sustainable PLM through intelligent products," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 2, pp. 789–799, 2013.

[17] S. Kubler, W. Derigent, K. Främling, A. Thomas, and E. Rondeau, "Enhanced product lifecycle information management using "communicating materials"," *Computer-Aided Design*, vol. (DOI) 10.1016/j.cad.2013.08.009, 2013.

[18] K. Främling, V. Hinkka, S. Parmar, and J. Tätilä, "Assessment of standards for inter-organizational tracking information exchange in the supply chain information control problems in manufacturing," in *Information Control Problems in Manufacturing*, 2012.