P2P Data Synchronization for Product Lifecycle Management

Sylvain Kubler[a,*], Kary Främling[a], William Derigent[b]

[a]*Aalto University, Department of Computer Science and Engineering*
P.O. Box 15400, FI-00076 Aalto, Finland
[b]*Université de Lorraine, CRAN, UMR 7039, Vandœuvre lès Nancy, F-54506, France*
CNRS, CRAN, UMR 7039, Vandœuvre lès Nancy, F-54506, France

Abstract

Intelligent products are an undeniable asset for efficient Product Lifecycle Management (PLM), providing ways to capture events related to physical objects at various locations and times. Today and more than ever before, PLM tools and systems must be built upon standards for enhancing interoperability among all product stakeholders and developing tools independent of specific vendors, applications, and operating systems. Based on this observation, this paper develops strategies to improve "information sustainability" in PLM environments using standardized communication interfaces defined by a recent standard proposal named Quantum Lifecycle Management (QLM) messaging standards. More concretely, data synchronization models based upon QLM standards are developed to enable the synchronization of product-related information among various systems, networks, and organizations involved throughout the product lifecycle. Our proposals are implemented and assessed based on two distinct platforms defined in the healthcare and home automation sectors.

Keywords: Product Lifecycle Management, Internet of Things, Intelligent products, Data synchronization, Multi-agent systems, Quantum Lifecycle Management

1. Introduction

The so-called "Internet of Things" (IoT) relies on the automatic capture of observations of physical objects at various locations, their movements between locations, sensor data collected from sensors attached to the objects or within their immediate surroundings, and interactions with people, mobile devices, other objects, and locations visited by the objects (and people and devices) at various times [1, 2, 3, 4]. This vision of the IoT is, to a certain extent, linked to the concept of Product Lifecycle Management (PLM) [5, 6], which is commonly understood as a strategic approach that incorporates the management of data associated with products of a particular type, and perhaps the versions and variants of that product type, as well as the business processes that surround it. It is a fact today that the advent of the IoT has radically reduced the communication cost for remote collaboration and coordination in PLM environments (in and between organizations), which leads to transcending the traditional organizational boundaries and space limitations [7, 8, 9]. These new boundaries reveal a real need for new types of interactions between various types of "things" (mobiles, sensors, computers, information systems, users, *etc.*) to help companies to deal with complex, changing product environments and to meet the new organizational and customer needs [10, 11]. In order to propose efficient PLM tools addressing all these challenges, it is a necessary condition to rely on standardized IoT interfaces rather than on domain-specific applications; the reason is two-fold: *i)* it enhances interoperability among all organizations/stakeholders involved in the product lifecycle [12], and *ii)* it provides more possibilities for developing new tools and models independent of specific vendors, applications, and operating systems [13, 14].

The Quantum Lifecycle Management (QLM) messaging standards have been developed and proposed as a standard that fulfill the main IoT requirements [15]. These standards provide generic and standardized interfaces for creating the needed information flows between all devices and systems that the IoT is composed of. The "QLM cloud" depicted in Figure 1 is intentionally drawn in the same way as the Web cloud; 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. Based on the QLM standards, this paper investigates new strategies for improving "information sustainability" in PLM systems. The main characteristic that makes

*Corresponding author
Email addresses: sylvain.kubler@aalto.fi (Sylvain Kubler), kary.framling@aalto.fi (Kary Främling), william.derigent@univ-lorraine.fr (William Derigent)

information "sustainable" for systems and users is that this information should be available whenever it is needed, wherever it is needed, and by whoever needs it, while being "consistent" (i.e., not outdated or wrong) [16, 17]. Information consistency can be maintained by implementing data synchronization mechanisms [18]. Accordingly, in more concrete level, this paper investigates and develops data synchronization models based upon QLM that make it possible to synchronize any product-related information among various and distinct lifecycle entities (e.g., distinct organizations, networks, information systems, devices...). The originality of these models lies in the fact that they do not require the addition of any software or hardware with QLM that significantly increases the scope of research and applications, and these models could be used in conjunction with any other messaging protocol (other than QLM) that provides the set of interfaces needed.

Section 2 provides both the IoT and data synchronization backgrounds from a PLM perspective. Section 3 provides insight into traditional data synchronization principles that ought to be integrated using the generic QLM interfaces. Such interfaces are introduced in section 4. Section 5 presents the generic data synchronization models that are developed based upon QLM. These models are then implemented and validated by considering two distinct platforms in the healthcare sector and home automation; the conclusions follow.

2. IoT and data synchronization background from a PLM perspective

PLM is a wide-ranging concept aiming to integrate nearly everything in the product lifecycle, including people, data, products, processes, metrics, and so forth. Section 2.1 provides an overview of PLM and related concepts. Existing strategies of data synchronization intended to be non vendor- and domain-specific (i.e., appropriate for PLM) are then reviewed in section 2.2.

2.1. Product Lifecycle Management (PLM)

As previously mentioned, PLM is commonly understood to be a strategic approach that incorporates the management of data associated with products [5, 6]. These product definition data are generated when the product is first conceived and then continue to evolve with the addition of detailed specifications, user manuals, computer-aided design (CAD) drawings, manufacturing instructions, service manuals, disposal and recycling instructions. Figure 2 provides insight into a traditional product lifecycle from the Beginning of Life (BoL), including the design, production and distribution of the product, through the

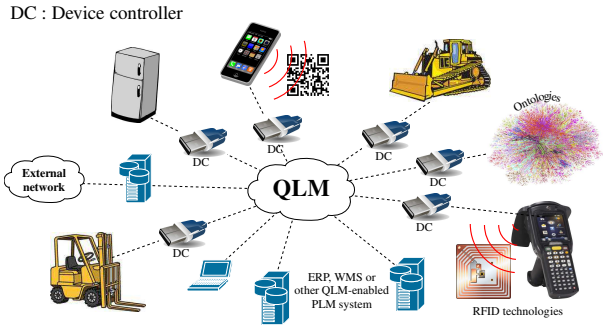


Figure 1: QLM "Cloud"

Middle of Life (MoL), including its use and maintenance, up to the End of Life (EoL), including its recycling. In traditional PLM, the product information generation process seems to end after the BoL. When the product enters actual use, PLM mainly provides access to existing information, but very little new information is generated about the products. This is, perhaps, a reflection of the point of view of the manufacturing industry that tends to see PLM mainly as a distributed knowledge management task of the "extended enterprise" [19] that created the product. Within this context, there has been only slight interest in how the customer uses each individual product or in how that product has behaved. Concepts such as "product agents" [20], "product-centric" PLM [21], and "intelligent products" [22, 23, 24] have been proposed as solutions for enabling such item- or instance-enabled PLM. Such concepts were the cornerstones of the product instance-enabled PLM solutions developed in the PROMISE EU FP6 project[1]. This project introduced the *closed-loop PLM*® paradigm [25, 26], recently renamed CL_2M (Closed-Loop Lifecycle Management), whose breakthrough challenge is to enable the information flow to include the customer and to enable the seamless transformation of information to knowledge. As a result, CL_2M and similar paradigms such as the "Closed-Loop Supply Chains" [27] or "Reverse Logistics" [28] contribute to enhancing various aspects of PLM, five of which are of the utmost importance:

1. *information security:* to maintain the level of security and confidentiality required by the organizations [29];
2. *information manageability:* to efficiently and properly process large amounts of raw data [10];
3. *information interoperability:* to manage the many changes in data media and formats throughout the product lifecycle and to ensure information exchanges between any kinds of products, users, and systems [30];

[1] http://promise-innovation.com

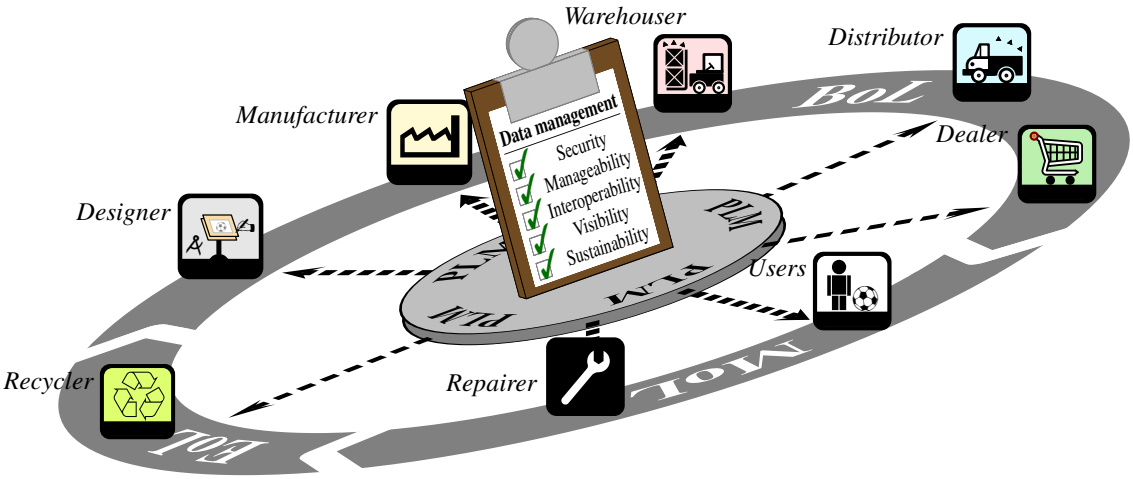


Figure 2: Traditional product lifecycle and major data management aspects to be addressed by a PLIM initiative

4. *information visibility:* to make data available for any system, anywhere, and at anytime. The CL_2M consortium[2] defines the visibility of the information as the possibility to gather, process, and exchange the desired information throughout the whole life of an entity [31, 32];

5. *information sustainability:* to make data capable of outliving systems, while being consistent [11].

These five aspects must be properly addressed for an efficient PLM, as highlighted in Figure 2. However, this is particularly challenging for organizations because PLM is a wide-ranging concept intended to manage the entire product lifecycle, considering all activities throughout which the product operates (from the BoL to EoL) [6]. Within this context, one can understand that it is of the utmost importance to develop and propose sufficiently generic approaches to address each of these aspects in all possible domains. This form of reasoning has been the starting point of the PROMISE consortium that created two main specifications aiming to improve *information interoperability* and *information visibility* throughout the product lifecycle, namely, the PROMISE System Object Model (SOM) and the PROMISE Messaging Interface (PMI). At the end of the PROMISE project, the work on these standards proposals was moved to the Quantum Lifecycle Messaging (QLM) workgroup of The Open Group[3]. QLM messaging standards are derived from the PMI and are intended to provide sufficiently generic and generally applicable IoT messaging standards to fulfill the main IoT requirements [15]. This is a necessary step for efficient PLM and to deal with all of the five aspects previously mentioned. In this paper, we investigate new ways to synchronize information throughout the product lifecycle making direct use of the communications interfaces defined in QLM, thus improving *information sustainability*.

2.2. Inter-organizational data synchronization

Since the 1980s, retailers and their suppliers have been a prominent example for electronic collaboration in the supply chain. Many surveys and studies showed that poor product data quality (e.g., outdated or wrong data) negatively impacts the benefits stakeholders pursue by implementing tighter forms of collaboration [33, 16, 34]. During the late 1990s and early 2000s, there was an increased awareness that the only viable and feasible solution was to develop appropriate standards-based processes that were globally acceptable to all involved parties [35]. Three of the main groups that were active in the early data synchronization efforts were the Global Commerce Initiative (GCI) that became part of the Consumer Goods Forum, the UCCnet® trading partner community that became part of the Global Data Synchronization Network[4] (GDSN), and the Auto-ID centers (founders of EPC – Electronic Product Code – information services that became part of GS1) [36, 37]. Each represented a community of many user members in different industry sectors and roles in the supply chain. GDSN is one of the most common examples of ecosystem synchronization that attempts to reduce inefficiencies in sharing standardized product data that hinder electronic supply chains, electronic markets, and *e*-collaboration [38]. Users upload product and location data to a registry, referred to as the "GS1 Global Registry", which allows trading partners to have access to updated, synchronized information. This solution therefore creates one location to allow multiple trading partners to

[2] http://www.cl2m.com
[3] http://www.opengroup.org/qlm/

[4] GDSN is associated to GS1 (for Global Standards): www.gs1.org

maintain updated supply chain data with one another. Such solutions have cleared up many issues related to product information management, especially for data synchronization in supply chain environments. Nonetheless, considering the entire product lifecycle, some limitations still need to be addressed. For instance, the limitations often do not allow for data synchronization/replication in a peer-to-peer, loosely coupled way (a centralized database, or registry, is always required), and the scope of action does not cover most of the IoT applications in the MoL (it is mainly limited to the supply chain, i.e. the BoL) [11].

Alongside these efforts, other groups and consortiums addressed similar issues in different contexts and environments such as the Synchronization Markup Language (SyncML) initiative, Pumatech's Intellisync [39], and Characteristic Polynomial Interpolation Synchronization (CPISync) [40]. The SyncML standard introduced by a consortium of companies in 2000 might be the best-known in the literature, which is now consolidated into the Open Mobile Alliance[5] (OMA) composed of a large number of companies (mobile operators, network suppliers, service providers…). The primary objective of OMA is to offer an open standard as a replacement for existing data synchronization solutions, which have mostly been vendor-, application-, or operating system-specific [41]. This standard defines an XML-based synchronization model that is portable and independent of specific data models and has the potential to handle data synchronization between distinct systems and mobile devices. Nonetheless, existing applications using OMA are still confined to particular domains such as healthcare applications [42, 43], automotive applications [44], or smart grid and cities [45, 46]. OMA was initially developed for synchronizing calendars, contact lists, and similar information, mainly between mobile phones and servers. Although it has been stated that OMA is suitable for peer-to-peer data synchronization [47], the synchronization model is closer to a client-server relationship established between one or more mobile devices and a server than a real peer-to-peer situation [41]. Given this observation, OMA does not perfectly comply with the real IoT philosophy in which client-server architectures and peer-to-peer data synchronization between devices with low processing capacity could be initiated, handled, and canceled at any time and for any type of data. Although OMA can be used conjointly with QLM messaging, it could be beneficial to propose solutions for data synchronization directly using standardized IoT interfaces, without adding any software or hardware that could prevent in some respects the use of other interesting features/interfaces proposed by the messaging protocol. For instance, unlike OMA, QLM messaging provides crucial features to address different types of IoT challenges such as *i)* the real-time discovery of information on any device, server, and information system that can therefore be synchronized at any time, or still *ii)* the possibility of achieving two-way "full-duplex" communications in a generic way through firewalls and with mobile systems (e.g., to enable real-time control and maintenance) [48]; see [15] for additional examples. Finally, if QLM standards could come to be used in numerous organizations and applications from different lifecycle phases and application-domains, the support of traditional data synchronization principles could lead to substantial time and cost savings.

[5] http://openmobilealliance.org

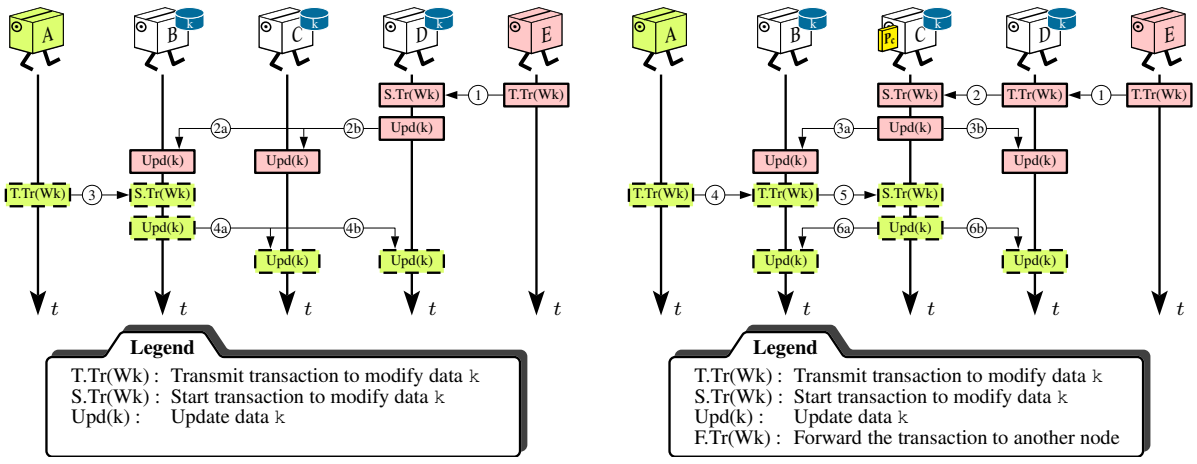
3. Data synchronization principles

Synchronizing data in any system requires two aspects to be considered: *i)* synchronizing the updated data among all nodes that carry a replica of that data and *ii)* handling the data access rights (i.e., to allow one or a group of nodes to modify it). The context of the application strongly influences the choice of the suitable data synchronization technique. Two dimensions must be considered:

- *When do the updates have to be propagated?* Two modes exist: Synchronous (S), for which it is necessary to inform every node carrying a replica that a modification will occur, and Asynchronous (A), which allows a node to carry out local modifications without informing its peers. The broadcasting of the updates is thus performed in a lazy way [49];
- *Where do the updates have to be performed?* Two principles exist: Primary copy (P_c), which allows one and only one node to modify the data, and Update everywhere (U_e), which enables one group of nodes to modify it.

Finally, four traditional techniques of data synchronization, also referred to as "data replication", can be defined[6] [50]: *i)* A-U_e, *ii)* A-P_c, *iii)* S-U_e, and *iv)* S-P_c. Each technique has *pros* and *cons,* as detailed in [51]. In the context of IoT as well as PLM, nodes are either fixed or mobile entities, with varying degrees of "intelligence" that can range from simple barcodes or RFID tags to vehicles and other products that have advanced sensing and actuating capabilities [9] (*cf.* Figure 1). Within this context, one can understand that the four techniques of data synchronization might be required by users and organizations according to their needs, activities, and infrastructures. As a first step,

[6] In order to remain consistent, the term "data synchronization" is used across the paper.



(a) Asynchronous - Update everywhere (A-U_e)

(b) Asynchronous - Primary copy (A-P_c)

Figure 3: Techniques of data synchronization

we investigate the feasibility of implementing the two asynchronous techniques (i.e., A-U_e and A-P_c) using the standardized QLM interfaces.

Figure 3 illustrates the behavior of the two asynchronous techniques considering five nodes: *A*, *B*, *C*, *D,* and *E*. Let k be a data replica carried by nodes *B*, *C*, and *D*. Considering the A-U_e technique, all nodes are allowed to asynchronously update k. In Figure 3(a), *E* sends a write request to *D* for modifying k (see communication/arrow denoted by "1"), which proceeds to perform the modification and updates replicas on *B* and *C* (see arrows denoted by "2a" and "2b"). Subsequently, *A* initiates a similar transaction addressed to *B* (see arrows denoted by "3"), which modifies k and updates replicas on *C* and *D* (see arrows denoted by "4a"/"4b"). The A-U_e technique may lead to conflicts between the different nodes because all nodes are allowed to modify k without consulting each other (e.g., a resulting action could be to reject the update or to raise an alert if a conflict occurs between two data replicas). Reconciliation mechanisms nonetheless exist to handle such conflicts [52, 53]. This problem is avoided when using the A-P_c technique (see Figure 3(b)) because only the node designated as P_c is allowed to modify k. Let node *C* be designated as P_c in Figure 3(b). Following the same transaction pattern as before, *E* sends a write request to *D* (see arrow denoted by "1"), which automatically forwards the request to the P_c node (i.e., *C*). Node *C* proceeds to perform the modification and then updates replicas on *B* and *D* (see arrows denoted by "3a"/"3b"). The same communication scheme appears when *A* addresses its transaction to *B* (see arrows denoted by "4", "5", and "6a"/"6b").

Because our ultimate goal is to develop strategies to support the four data synchronization previously outlined using only the generic interfaces defined by the QLM standards, a high-level description of those interfaces/standards is given in section 4.

4. QLM messaging standards

QLM messaging specifications consist of two standards proposals [15]: 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. In the QLM cloud, communication between the participants (e.g., products and backend systems) is performed by passing messages between nodes using QLM-MI. 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 due to its flexibility, which provides more opportunities for complex data structures [10], but others such 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. QLM-MI and QLM-DF are independent entities that reside in the Application layer of the OSI model, where QLM-MI is specified at the Communication level and QLM-DF is specified at the Format level.

4.1. 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

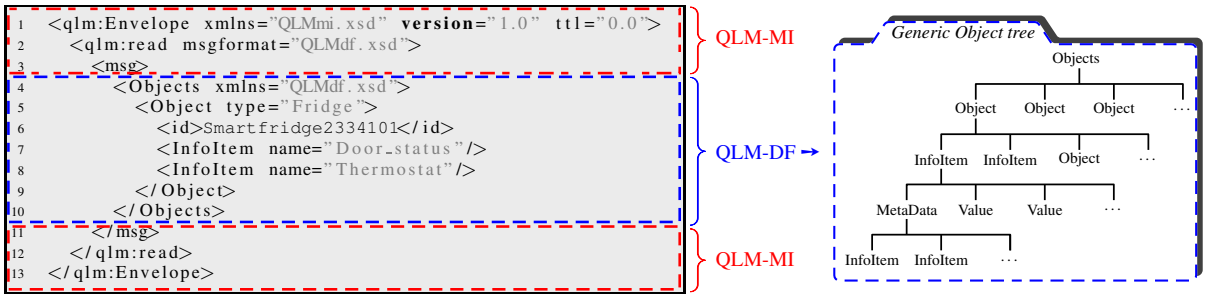


Figure 4: QLM Data Format: generic "Object" tree and example of a QLM message relying on that tree

as a hierarchy with an "Objects" element as its top element. The "Objects" element can contain any number of "Object" sub-elements. Figure 4 gives insight into both the generic hierarchy/object tree and an example of a QLM message whose structure respects this object tree. In this example, a unique object of type `Fridge` (see row 5 of the XML message) is considered. "Object" elements can have any number of properties, referred to as InfoItems, as well as "Object" sub-elements. In our example, the Object `Fridge` has two InfoItems named `Door_status` and `Thermostat` (see rows 7 and 8). The resulting Object tree can contain any number of levels. Every Object has a compulsory sub-element called "id" that identifies the Object (see row 6). The "id" should preferably be globally unique or at least unique for the specific application, domain, or network of the involved organizations.

4.2. 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. Three operations are possible:

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

1. *subscription **with** callback address:* the subscribed data are sent to the callback address at the requested interval. Two types of intervals can be defined: *interval-based* or *event-based*;
2. *subscription **without** callback address:* the data are 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.

Another important feature of QLM-MI is that QLM messages are "self-contained" in the sense that all the necessary information (e.g., the actions to be performed, the callback address...) to enable the recipient to handle the message is contained in the message itself. The QLM message given as example in Figure 4 highlights the message interface in which the TTL (time-to-live) value is specified in row 1 and the operation type is specified in row 2 ("read"). Other relevant interfaces are presented in [15] such as the "publication and discovery" of new data sources, services and meta-data by using "RESTful" URL-based queries (including discovery by search engines).

Section 5 investigates how asynchronous data synchronization techniques can be properly integrated based upon QLM messaging, i.e., by making direct use of both the interfaces defined in QLM-MI and the Object tree defined in QLM-DF.

5. Strategy for asynchronous data synchronization using QLM

The strategy defined in this paper to integrate data synchronization techniques based upon QLM uses the concept of "Product agent" [54, 55, 20]. In computer science, an agent is a piece of software acting autonomously; it has its own data structures and decision-making algorithms. For decades, agent platforms have been used in multiple research domains (sociology, chemistry, biology...) because they help to realize important properties such as autonomy, responsiveness, redundancy, distributedness, and openness in the system [56]. An agent may operate in environments where other agents operate, with the whole operating as a multi-agent system [57, 58].

In our approach, a QLM node that is intended, or should be involved in a data synchronization

must have an agent in charge of initiating, handling, and removing (if necessary) the synchronization in the whole system by collaborating with other QLM nodes. Such an agent is referred to as a *DS_agent* (D and S being the initials of Data Synchronization) and each of its options (i.e., "initiate", "handle", and "remove" synchronization) are detailed in sections 5.1 and 5.2.

5.1. "initiate" and "remove" synchronization

Options *initiate()* and *remove()* provide the user with parameters to set up or cancel a data synchronization among a set of QLM nodes for a specific InfoItem. These two options are described in the same section because they both depend on inputs specified by the user. We refer to these options as *externally based inputs*, as emphasized in Figure 5. Three parameters must be specified (using a GUI), as depicted in cells *a* and *b*:

1. the name of the InfoItem, denoted by II_{n}, whose replicas have to be handled in the QLM cloud;
2. the list of QLM nodes, denoted by L_n, that carry a replica of II_{n} and that are affected by the creation or cancellation of a data synchronization. It is required that all QLM nodes in L_n implement the *DS_agent*;
3. the technique of data synchronization, denoted by *tech*, that has to be implemented or removed among the QLM nodes in L_n. If A-P_c is selected, the P_c node must be designated.

Algorithm 1: `initiate(`II_{n}, L_n, *tech*`)`

output: $\mathcal{M}_{in}$

```
1 begin
2     M_in ← ∅; // Initialize the set of messages to be sent to empty
3     obj ← DS init; // Set the type of the "Object" element of the message to "DSinit"
4     forall the @_j ∈ L_n do // for all nodes ∈ L_n
5         M =qlmWrite(obj, II_n, L_n, tech, @_j);
          // Create the appropriate QLM write message according to the specified inputs
6         M_in ← M_in ∪ M; // Add message to M_in
```

The *DS_agent* embedded in the QLM node requesting the creation or removal of a data synchronization must communicate the necessary information to all nodes $\in L_n$. Algorithms 1 and 2 provide the respective functions to generate the right number of QLM messages that all contain the three parameters previously described; message are denoted by "✉ qlm(*in.*)" in cell *a* (*in.* being the abbreviation for *initiate*) and

Algorithm 2: `remove(`II_{n}, L_n, *tech*`)`

output: $\mathcal{M}_{re}$

```
1 begin
2     M_re ← ∅;
3     obj ← DS rem; // Set the type of the "Object" element of the message to "DSrem"
4     forall the @_j ∈ L_n do
5         M =qlmWrite(obj, II_n, L_n, tech, @_j);
6         M_re ← M_re ∪ M;
```

by "✉ qlm(*re.*)" in cell *b* (*re.* being the abbreviation for *remove*) depending on the option selected. The sub-function `qlmWrite()` (see row 5 in Algorithm 1 and 2) is in charge of generating the appropriate XML message *M* according to the specified function inputs (obj, II_{n}, L_n, *tech*, $@_j$), whose message must be sent to QLM node $@_j$. When node $@_j$ receives *M*, it is thus able to know how to process the message. For instance, considering node B in cell *c* of Figure 5, it knows that *M* (i.e., ✉ qlm(*in.*)) is a request for the establishment of a data synchronization of type *tech* regarding InfoItem II_{n}, which needs to be set up between nodes $\in L_n$. The logic is similar when removing a data synchronization (*cf.* cell *e*).

Let us consider a concrete scenario when the user initializes a data synchronization from node A (see cell *a*), proving the three following parameters:

- II_{n} = `Temperature1326`;
- *tech* = A-P_c with node C designated as P_c;
- L_n = {192.168.1.3, 192.168.1.1, 192.168.1.2} that are the respective IP addresses of nodes C, A, and B in Figure 5; a convention is defined to indicate which node is designated as P_c that consists of positioning the P_c node address at the first position in L_n.

Using Algorithm 1, the *DS_agent* embedded in A generates three QLM messages of type "✉ qlm(*in.*)" including the three parameter values. These messages are respectively sent to nodes B, C, and D (see arrows between cells *a* and *c* in Figure 5). One of these messages is given in Figure 6, in which a unique Object of type *DSinit* is defined (see row 5) and used to integrate the three parameters. These parameters are included using three distinct InfoItems named "IIn" (see row 7), "tech" (see row 10), and "Ln" (see row 13). The optional sub-element "Value" is used to indicate each parameter value (see rows 8, 11, and 14-16). The process is similar for the *remove()* option (see arrows between cells *b* and *e* in Figure 5), except that the "Object type" is set to *DSrem* and not *DSinit*, which enables the recipient node to distinguish both options and to act accordingly.

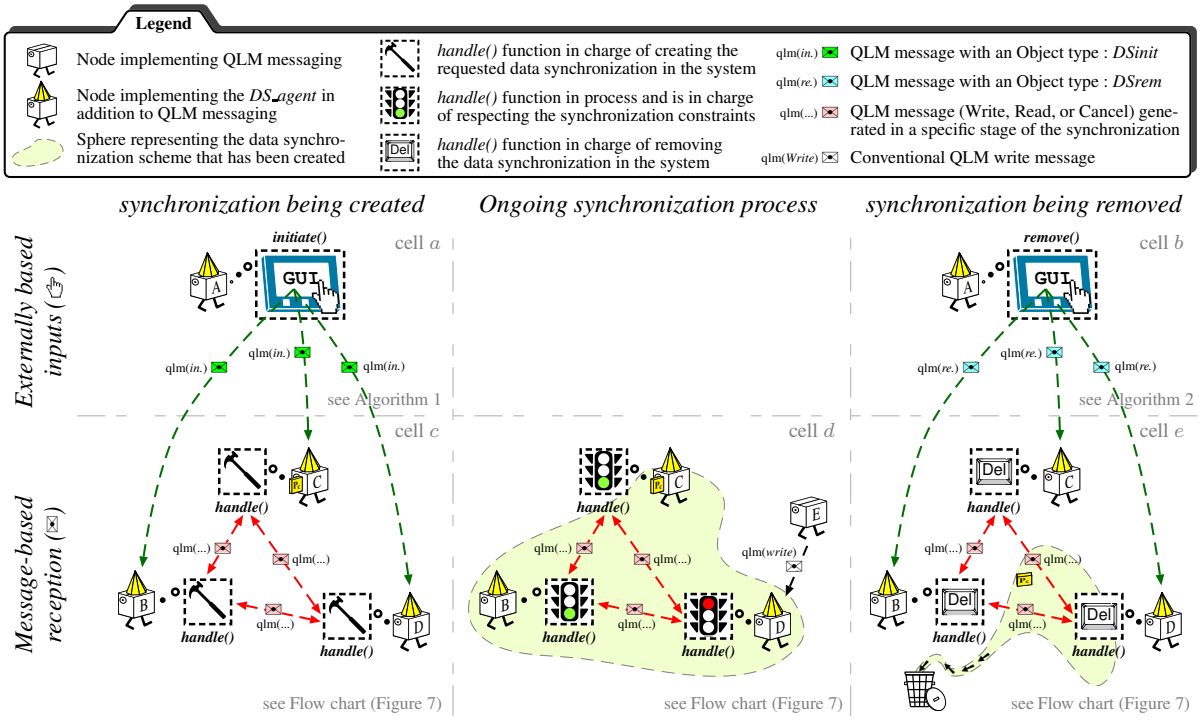


Figure 5: *DS_agent* and its three constituent options

```
<qlmEnvelope xmlns="QLM_mi.xsd" version="1.0" ttl="10">
  <write msgformat="QLM_df.xsd">
    <msg>
      <Objects xmlns="QLM_df.xsd">
        <Object type="DSinit">
          <id>DSinit13</id>
          <InfoItem class="IIn">
Parameter 1 <value>Temperature1326</value>
          </InfoItem>
          <InfoItem class="tech">
Parameter 2 <value>A-Pc</value>
          </InfoItem>
          <InfoItem class="Ln">
            <value type="csv">
Parameter 3 192.168.1.3,192.168.1.2,192.168.1.2
            </value>
          </InfoItem>
        </Object>
      </Objects>
    </msg>
  </write>
</qlmEnvelope>
```

QLM-MI / QLM-DF / QLM-MI

Figure 6: ✉ qlm(*in.*) including the 3 parameters defined by the user

The set of QLM nodes (L_n) must then start to implement (or remove) the data synchronization in the whole multi-*DS_agent* system based on the information contained in messages it receives or, to be more accurate, based on the set of parameters contained in these messages. Such a synergy is made possible thanks to the *handle()* option that is described in the next section. This option corresponds to cells *c*, *d*, and *e* in Figure 5 depending on the type of message received by the QLM node, which can be either of type "✉ qlm(*in.*)" (see cell *c*), of type "✉ qlm(*re.*)" (see cell *e*), or of any other type, i.e. any message that contains "Object" types different from *DSinit* or *DSrem* (corresponds to cell *d*). Unlike *initiate()* and *remove()* options, *handle()* is referred to as *message-based reception* because it all starts with the reception of a QLM message, as shown in Figure 5.

5.2. "handle" synchronization

This option is considered the core of the *DS_agent* because it is in charge of creating synergy in the multi-*DS_agent* system for initiating, handling, and even removing a data synchronization according to the incoming messages. More concretely, the *handle()* option is in charge of:

i) creating the appropriate subscriptions in the multi-*DS_agent* system when initiating a synchronization;

ii) respecting the synchronization constraints during the period of validity of that synchronization;

iii) canceling the subscriptions that have been set up in the multi-*DS_agent* system when removing a synchronization.

An accurate view of this option is provided in the form of a flow-chart in Figure 7. First, the function identifies what QLM operation is requested by the incoming message (see box connector denoted by "1" in Figure 7). If it is a "Read", "Cancel", or "Response" message, it is handled in the classic way (i.e., no further actions than the ones defined in the QLM specifications are required). Accordingly, the message is directly transmitted to the middleware that integrates

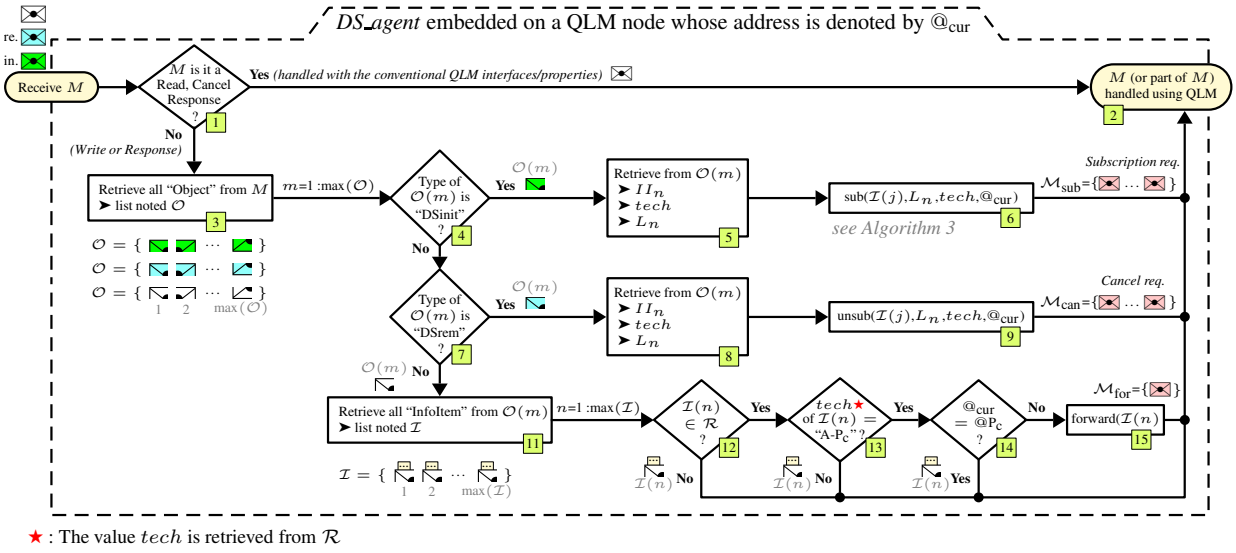


Figure 7: Flow chart describing the `handle()` function of the *DS_agent*: Message-based reception (see Figure 5)

these specifications (see box connector "2"). If it is a "Write" message, the function has to further investigate whether it contains an Object of type *DSinit* (see box connector "4"), *DSrem* (see box connector "7"), or of other types (see box connectors "11"). Sections 5.2.1, 5.2.2, and 5.2.3 respectively detail the actions to be undertaken by the *DS_agent* for each of these object types.

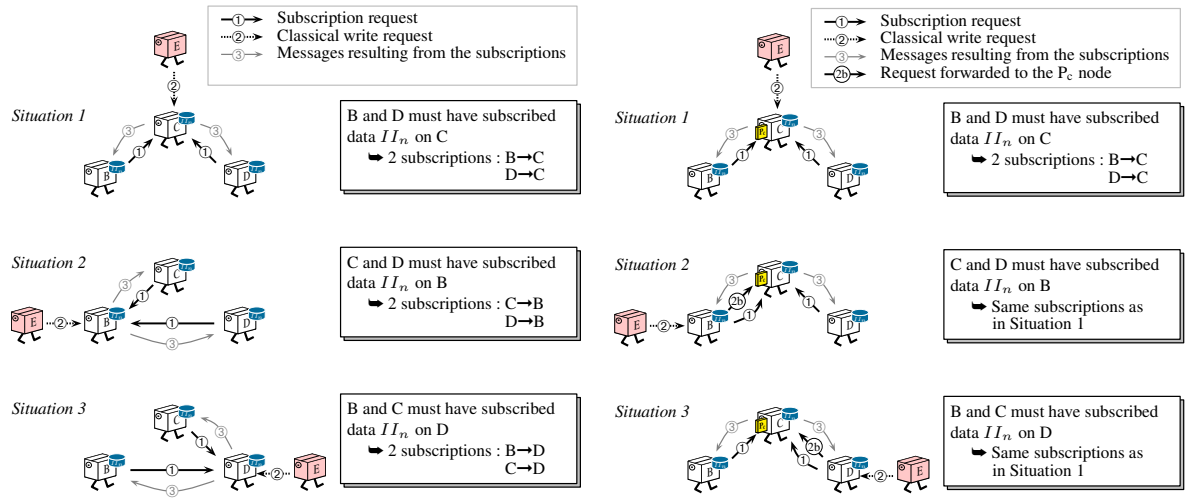
5.2.1. *Object – DSinit*

When a node receives a message M containing an "Object" of type *DSinit*, the set of InfoItems that compose this object (i.e., II_n, *tech*, and L_n; *cf.* Figure 6) are first retrieved as indicated in the box connector denoted by "5" in Figure 7. Based on these InfoItem values, it is then necessary to set up the appropriate number of subscriptions among the QLM nodes in order to push the II_n value to all nodes that carry a replica each time II_n is modified. The number of subscriptions to be set up depends on the selected principle (i.e., P_c or U_e).

Because the U_e principle allows all nodes carrying a replica to modify it, $z(z-1)$ subscriptions must be originally defined in the system, with z the number of nodes in L_n (i.e., each node subscribes to each other). In order to understand this equation, let us consider the example given in Figure 8(a) where nodes B, C, and D carry a replica of II_n. Three situations could potentially occur: a client could request to modify II_n on node C (situation 1 in Figure 8(a)), on node B (situation 2), or on node D (situation 3). Considering situation 1, nodes B and D must have beforehand subscribed to II_n on node C (see arrows denoted by ① in Figure 8(a)) by defining an *event-based* subscription and by providing their respective address as a *callback*. Hence, each time II_n is modified on node C (see arrow denoted by ②), all subscribers receive a notification with its new value (see arrows denoted by ③). The same occurs in the second and third situations. In total, when applying the generic formula with $z = 3$:{A, B, C}, $3(3-1) = 6$ subscriptions must be originally defined as indicated in Figure 8(a) (B→C; D→C; C→B; D→B; B→D and C→D). Figure 9 shows one of these 6 subscription requests that is, in this example, generated by the *DS_agent* embedded in node B and sent to node C to subscribe to InfoItem `Temperature1234` (see rows 7-9 of the request). Note that the interval parameter of the subscription (see row 2) is set to "-1", which indicates that it is an "event-based" subscription. Figure 9 highlights that each time `Temperature1326` is modified, a QLM response is automatically returned to the subscriber, including both the new II_n value and the ID of the subscription (denoted by ID_{sub} in Figure 9).

Because the P_c principle only authorizes the node designated as P_c to modify II_n, only $(z-1)$ subscriptions must be originally defined (all nodes subscribe to the P_c node, except the P_c node itself). This is illustrated in Figure 8(b) considering the same three situations, except that node C is now designated as P_c. Applying this formula with $n = 3$:{A, B, C}, $(3-1) = 2$ subscriptions must initially be defined (B→C and D→C). Although less subscriptions are required when using the P_c principle, an additional constraint must be handled by the *DS_agent*, namely, to forward the request to the node designated as P_c (see arrows denoted by "2b" in situations 2 and 3).

The appropriate number of subscriptions to be defined in the multi-*DS_agent* system, whether when selecting A-P_c or A-U_e, is computed using the function named `sub()` in box connector "6" in Figure 7, which is detailed in Algorithm 3 (see comments provided in



(a) Update everywhere (U_e) (b) Primary copy (P_c)

Figure 8: Number of subscriptions to be defined according to the selected principle: U_e or P_c

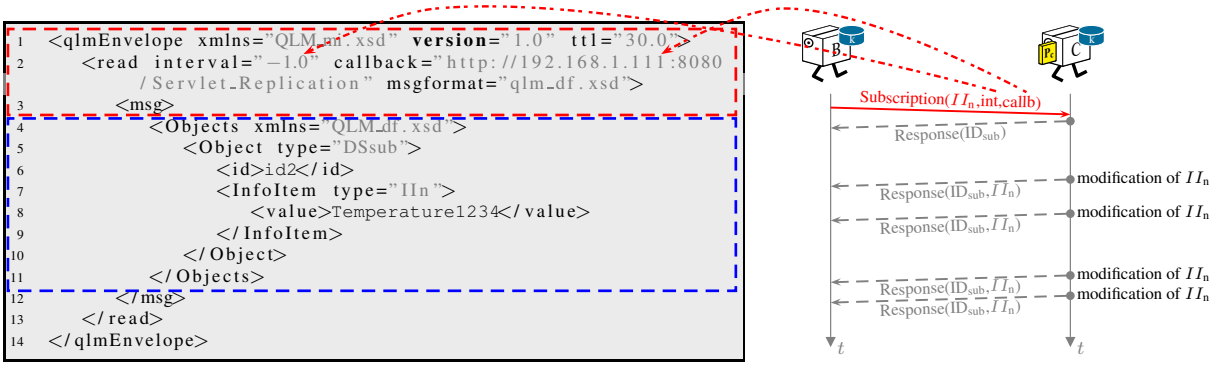


Figure 9: Generation of one subscription message *M* related to the `update` function

this algorithm for more details).

5.2.2. *Object – DSrem*

When a *DS_agent* receives a message *M* with an "Object" of type *DSrem*, the set of InfoItems that compose this object (i.e., II_n, *tech*, and L_n) are first retrieved (see box connector denoted by "8" in the flow chart). Based on these InfoItem values, it is then necessary to cancel the set of subscriptions that have been set up among the QLM nodes beforehand (regarding II_n). This is achieved using the `unsub()` function, as indicated in the box connector denoted by "9". This function is not presented in this paper but is similar to `sub()` (i.e., Algorithm 3), except that the generated messages are QLM cancel requests and not QLM subscription (read) requests.

5.2.3. *Object – other types*

When a *DS_agent* receives a Write message *M* with an Object type different than *DSinit* and *DSrem*, it first retrieves all InfoItems that compose this Object. This list is noted by $\mathcal{I}$ in box connector '11". For each InfoItem $n \in \mathcal{I}$, the function checks whether it is affected by a data synchronization, i.e. whether it has been registered in $\mathcal{R}$ or not (see box connector denoted by "12"). If not, no further actions are required, and the modification of InfoItem $\mathcal{I}(n)$ can be carried out using QLM standards (see connector between boxes "12" and "2"). If $\mathcal{I}(n)$ has to be synchronized, it is thus necessary to check whether the synchronization that has been established is A-P_c or A-U_e. This corresponds to the box connector denoted by "13" with *tech* being the technique of synchronization. In case it is A-U_e, no further actions are required, and the modification of $\mathcal{I}(n)$ can be carried out using QLM (see connector between boxes "13" and "2"). Otherwise, it is necessary to check whether the current node (i.e., $@_{cur}$) has been designated as P_c or not (see box connector denoted by "14"). If true, no further actions are required, and the modification of $\mathcal{I}(n)$ is carried out using QLM standards; otherwise, $\mathcal{I}(n)$ has to be forwarded to the P_c node, which is achieved thanks to the function named *forward()* in box connector "15".

Algorithm 3: sub($II_n, L_n, @_{cur}, tech$)

output: $\mathcal{M}_{sub}$

```
1  begin
2      M_sub ← ∅;                                   // The set of subscription requests M_sub is initialized to empty
3      callb ← @_cur;          // The callback address of the subscription corresponds to the QLM node that received M
4      inter ← −1;               // The interval is set to "-1" to indicate that it is an "event-based" subscription
5      if tech = "A-U_e" then                       // If the requested data synchronization is equal to "A-U_e, then"
6          forall the @_j ∈ L_n|@_j ≠ @_cur do          // for all nodes that carry a replica of InfoItem II_n
7              M =qlmRead(II_n, inter, callb, @_j);   // Send a subscription request to subscribe InfoItem II_n,
                considering the interval and callback values specified at rows 3 and 4
8              M_sub ← M_sub ∪ M;                   // Add message M to the set of subscription requests to be sent
9      else                                         // If the requested data synchronization is A-P_c
10         if @_cur ≠ @_Pc then                     // If the current node is not the designated P_c node, then
11             M =qlmRead(II_n, inter, callb, @_Pc);  // Send a subscription request to subscribe II_n to P_c
12             M_sub ← M_sub ∪ M;                   // Add message M to the set of subscription requests to be sent
13     R ← {II_n, L_n, tech};                       // The initiated data synchronization is memorized on the current node
```

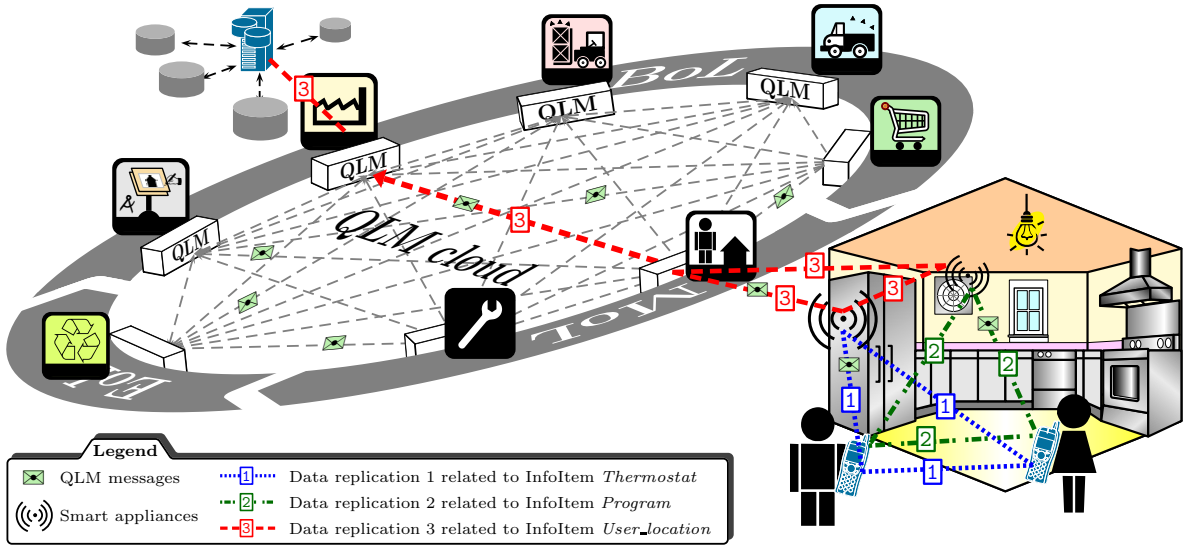


Figure 10: Home automation platform in which some lifecycle entities implement the *DS_agent* in addition to QLM messaging

6. Case study

Two platforms respectively defined for home automation and healthcare assistance are presented in sections 6.1 and 6.2. The different organizations and actors of these platforms implement the QLM messaging standards, and the *DS_agent* is used to manage information replicas generated among these actors. Integration of the QLM standards and the *DS_agent* is made possible in both case studies using the DIALOG middleware[7].

6.1. Home automation

The home automation environment consists of two smart household appliances, namely a *fridge* and a *ventilator*, which both integrate a controller. *Ensto eSmart*® controllers are used in our case study. Two residents (man and woman) are able to control these appliances via their smartphone. These four devices (2 smart appliances and 2 smartphones) are augmented with QLM capabilities and thus are able to communicate with each other using conventional QLM interfaces. Figure 10 illustrates this environment (in the MoL) as well as the different actors and devices. In our scenario, the fridge manufacturer also supports QLM to exchange lifecycle events and information about his own assets (e.g., fridges manufactured by his company) with all product stakeholders. This is made possible through the QLM cloud, as depicted in Figure 10, which interconnects all phases and actors involved in the fridge lifecycle.

[7] http://dialog.hut.fi

Table 1: List of data synchronizations set up between the house devices

	InfoItem (appliance)	Functionality and permissible values	Man's mobile	Woman's mobile	Manufacturer DB	Fridge controller	Ventilator controller	Data synchronization
1	*Thermostat* (Fridge)	Enable to adjust the indoor temperature of the fridge. Six thermostat values $\{1, 2, .., 6\}$ can be selected	✓	✓		✓		A-P_c ↣ P_c=fridge
2	*Program* (Ventilation)	Enable to select a ventilation program (13 in total)	✓	✓			✓	A-P_c ↣ P_c=ventilation
3	*eSmart_modes* (Fridge+Ventilation)	Enable users to indicate whether they are at $\{Home, Safe\ at\ home, Away, LongAway\}$			✓	✓	✓	A-U_e

Particular InfoItems on each appliance can be accessed and controlled using QLM. Table 1 details these InfoItems by indicating which appliance they are related to (see column 1), what the permissible values are (see column 2), and who is accessing these InfoItems. For instance, both parents are allowed to modify the fridge *Thermostat* as well as the ventilation *Program* (see rows 1 and 2 in Table 1). In both situations, a change of the InfoItem value on the controller (whether remotely or locally initiated) has to be updated to all devices that could potentially control it (i.e., the parent mobiles). Let us note that the modification needs to first be performed by the controller and then propagated in the system/QLM cloud. Accordingly, the A-P_c technique is selected by designating the controllers (fridge and ventilator) as P_c (see rows 1 and 2 in Table 1). The last InfoItem, denoted by *eSmart_modes*, is different insofar as it is present on both controllers; in other words, it is not unique in the QLM cloud. This InfoItem is used to control the status of the premises, depending on whether they are present or absent. Four modes are available to indicate whether they are at *Home*, *Safe at home* (i.e., sleeping), *Away*, or *Long away* (*cf.* Table 1). Figure 11 shows these four modes on one of the two *Ensto eSmart*® controllers. Different types of actions are therefore undertaken according to the current mode. For instance, some of the rules defined in the *Ensto eSmart* technical documentation are: *i)* alarms are inactive if residents are at *Home* and active if they are *Safe at Home*, *ii)* power from the electrical sockets in the kitchen is "off" if residents are *Away*, *etc.* These modes can be changed both locally (operating panel, button) and remotely (text message, call). However, when more than one controller is used in the same building, there is no mechanism in place to synchronize the same InfoItem located on two or more distinct controllers. Because the InfoItem *eSmart_modes* can be modified (locally or remotely) by the two residents on any of the two controllers (fridge or ventilator), the A-U_e technique is selected (see Table 1). In our platform, the premises agreed that the fridge manufacturer can also access specific InfoItems on the fridge controller (e.g., to take proactive actions

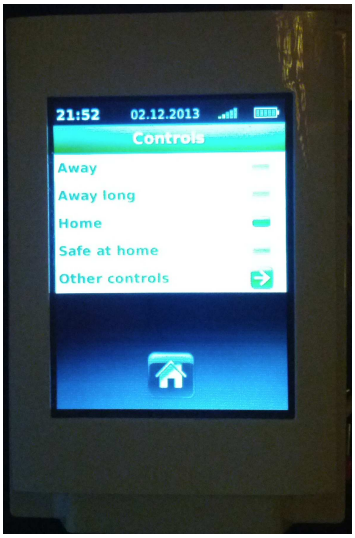


Figure 11: *Ensto eSmart*® controller with four presence modes

like the scheduling of a time for service, orders for needed spare parts). In this scenario, the manufacturer is interested in knowing when residents are *Away* or *Long away* from the house so that certain tests and updates can be performed that could require hours or even days. Accordingly, the manufacturer company database system is also included into data synchronization 3, as shown in Table 1 and illustrated in Figure 10 through communications denoted by "3"..

The *DS_agent* developed in this paper is used in addition to QLM in order to realize the three expected data synchronizations (*cf.* Figure 10). Section 6.1.1 details the steps resulting from such an implementation considering synchronizations 1 and 2 (of type A-P_c), while section 6.1.2 details these stages considering synchronization 3 (of type A-U_e).

6.1.1. Data synchronization 1 and 2 (A-P_c)

A screenshot of the GUI used to initiate (or remove) a data synchronization is shown in Figure 12. This figure shows the configuration of data synchronization 1: InfoItem *Thermostat* is selected, IP addresses of the two resident phones and the fridge are specified, and data synchronization of type A-P_c is selected (IP ad-

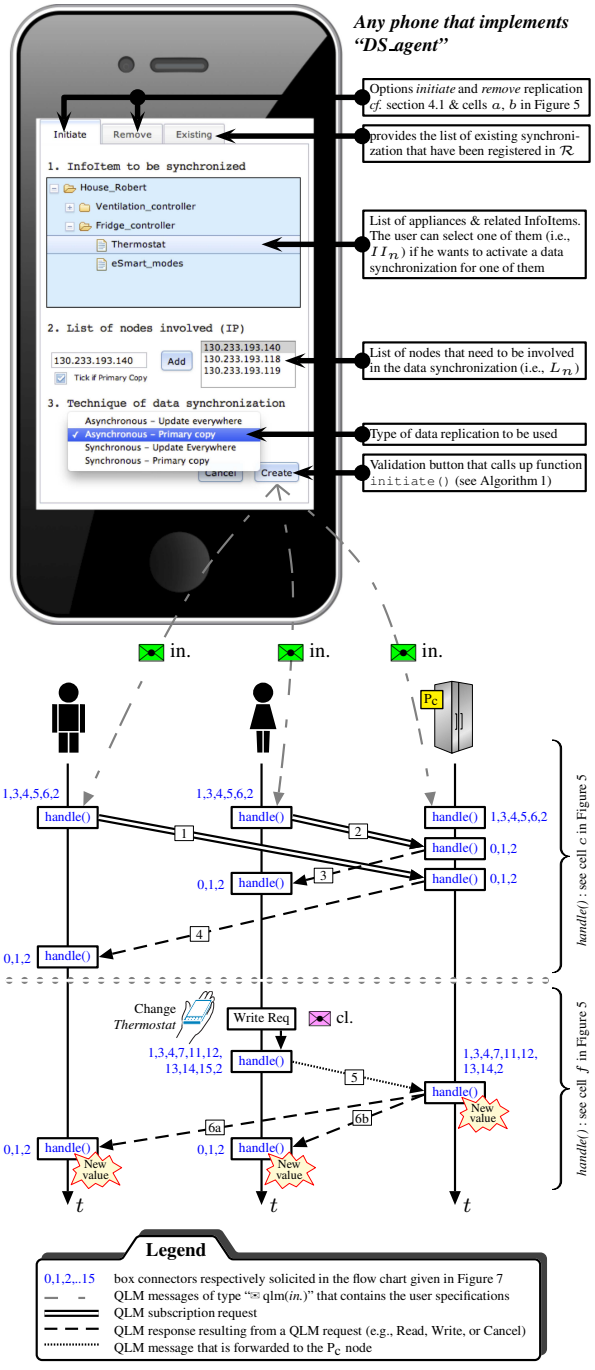


Figure 12: Data synchronization 1: GUI and sequence diagram

dress of the fridge is ticked as P_c). When clicking on the "create" button, the *initiate()* function is called (see Algorithm 1), leading to three QLM messages of type "✉ qlm(*in.*)", which are addressed to both residents and the fridge, as illustrated in Figure 12.

Each QLM message is automatically processed by the recipient QLM nodes using the *handle()* function integrated into the *DS_agent*, as highlighted in Figure 12. Because the received messages are of type "✉ qlm(*in.*)", box connectors 1, 3, 4, 5, 6, and 2 from the flow chart in Figure 7 are solicited. Following the explanations given in Figure 8(b), only $(3-1) = 2$ subscriptions are defined, namely, the two resident smartphones that subscribe to InfoItem *Thermostat* on the fridge (see arrows denoted by "1" and "2" in Figure 12). The fridge receives both subscription requests and thus solicits box connectors 0, 1, and 2 in Figure 7 because a subscription is a special QLM read request. After having created both subscriptions, the DIALOG middleware embedded in the fridge returns the subscription IDs to both subscribers (see arrows denoted by "3" and "4" in Figure 12). At that time, data synchronization 1 is ready for use.

Later, the woman requests to modify the *Thermostat* value, as depicted in Figure 12 (see box denoted by "Write Req"). The *handle()* function is therefore solicited to find out how to process such a request. Box connectors 1, 3, 4, 7, 11, 12, 13, 14, 15, 2 are solicited in the flow chart (see Figure 7) because *Thermostat* is involved in an active synchronization of type A-P_c and the woman's phone has not been designated as the P_c node. Accordingly, the request is automatically forwarded to the fridge, as depicted by arrow "5" in Figure 12. The fridge in turn solicits the *handle()* function, which is allowed to modify *Thermostat*. Because data synchronization 1 created the appropriate subscriptions, the new *Thermostat* value is automatically pushed (via a QLM response) to both resident phones, as shown in Figure 12 (see arrows "6a" and "6b" respectively).

Similarly, data synchronization 2 (InfoItem *Progam*) is set up to include the two residents and the ventilator controller.

6.1.2. Data synchronization 3 (A-U_e)

Data synchronization 3 is similarly created, as illustrated in Figure 13; three QLM messages of type "✉ qlm(*in.*)" are respectively sent to the manufacturer, the fridge, and the ventilator. As before, each QLM message is automatically processed by the recipient nodes using *handle()* (see Figure 13). Following the explanations given in section 5.2, $3(3-1) = 6$ subscriptions are now defined between the three actors (each actor subscribes to each of the others), as illustrated in Figure 13 (see the legend in Figure 12 for more information). A network protocol analyzer has been used to capture the set of QLM messages sent and received by the fridge (see Figure 13). It can be observed that the fridge first sends a subscription request to the Ventilator ("1a"), which responds back via a QLM response including the ID of the created subscription ("1b"). Similarly, the fridge sends a subscription request to the manufacturer system ("2a"), which also returns the ID of the created subscription ("2b"). Then, the fridge receives the requests sent by the ventilator ("3a") and the manufacturer ("4a"). The fridge creates the requested subscription (i.e., regarding InfoItem *eSmart_modes*) and returns to both

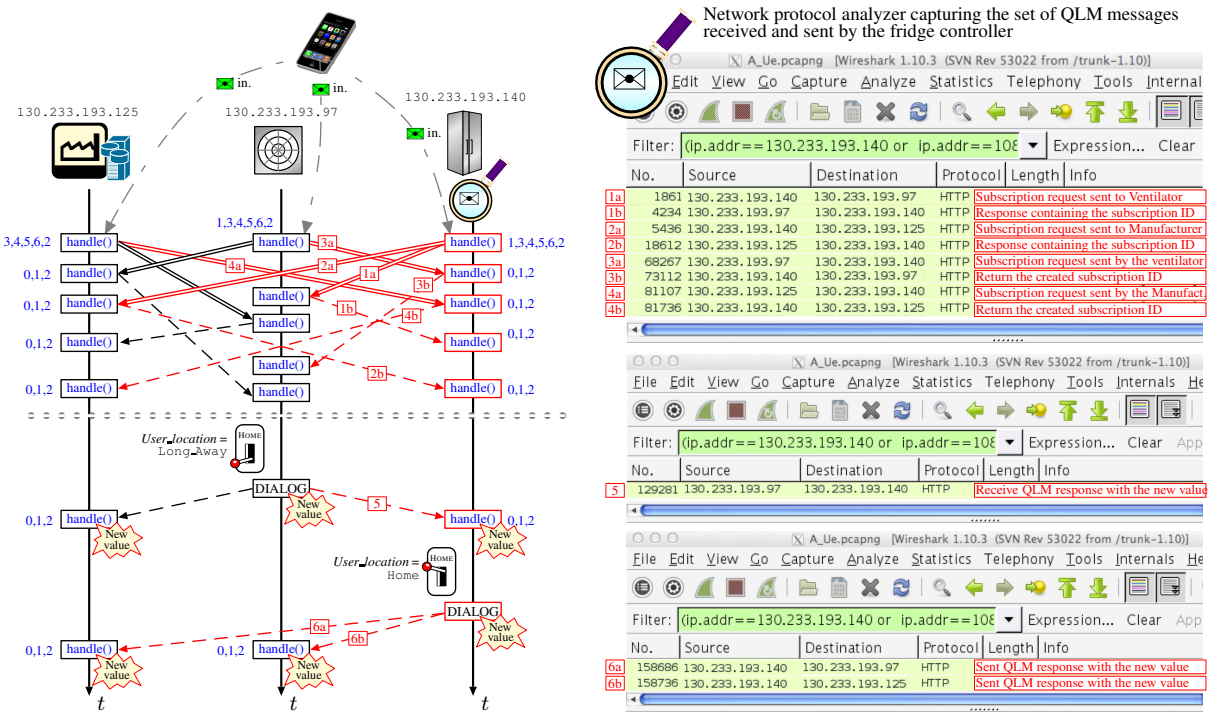


Figure 13: Sequence diagram and network protocol analyzer screenshots related to data synchronization 3 (A-U_e)

subscribers a response containing the subscription ID generated by DIALOG ("3b" and "4b"). At that time, data synchronization 3 is ready for use.

Later, the residents leave home for two weeks and, accordingly, switch the *eSmart_modes* interruptor on the ventilator from *Home* to *Long_Away* (see Figure 13). Because data synchronization 3 created the appropriate subscriptions, the new *eSmart_modes* value is automatically pushed (via a QLM response) to both the fridge and the manufacturer system as illustrated in Figure 13 (see arrow "5" and the network protocol analyzer). This synchronization avoids having a different mode selected on each controller and therefore avoids having decisions being made that have opposite effects (e.g., one controller trying to cool the house while another tries to heat it because their statuses are not synchronized to the same value). The synchronization also enables the manufacturer to be notified about the non-presence of the residents. When the residents come home, they switch the *eSmart_modes* interruptor from *Long_Away* to *Home,* as depicted in Figure 13, but this time using the fridge controller. The sequence diagram as well as the network protocol analyzer highlights the fact that data synchronization 3 generates two QML responses, which are sent by the fridge (DIALOG to be exact) to the ventilator and manufacturer systems (see arrows "6a" and "6b").

This case study enabled validation of the two asynchronous techniques (A-P_c; A-U_e) using the standardized QLM interfaces. The next section provides a similar case study but for health assistance purposes.

6.2. *Health assistance*

The healthcare scenario presented in this section considers a basic activity of daily living, namely "jogging". The jogger wears a watch able to monitor different features of his body and environment, such as the Heart Rate Variability (HRV), the muscle activity (electromyography – EMG), distances traveled, and other terrain features. A classic sport watch (Garmin Forerunner 620) is used in this case study to monitor both the jogger's HRV and the jogging conditions (weather events and elevation). When the jogger goes home, such data is wirelessly transferred to his computer, as shown in Figure 14. Three additional actors are considered:

- *the jogger's cardiologist:* the jogger agreed that the cardiologist can access particular information contained in the jogger's database, such as the HRV values and jogging conditions;
- *the jogger's neurologist and insurance company:* the jogger agreed that the cardiologist can communicate information related to the jogger's health to his neurologist and healthcare insurance company. It is becoming common for insurance companies to create financial incentives for people who voluntarily share information about their health (e.g., in US) [59, 60].

Only the cardiologist is allowed to retrieve information from the jogger database and to communicate it

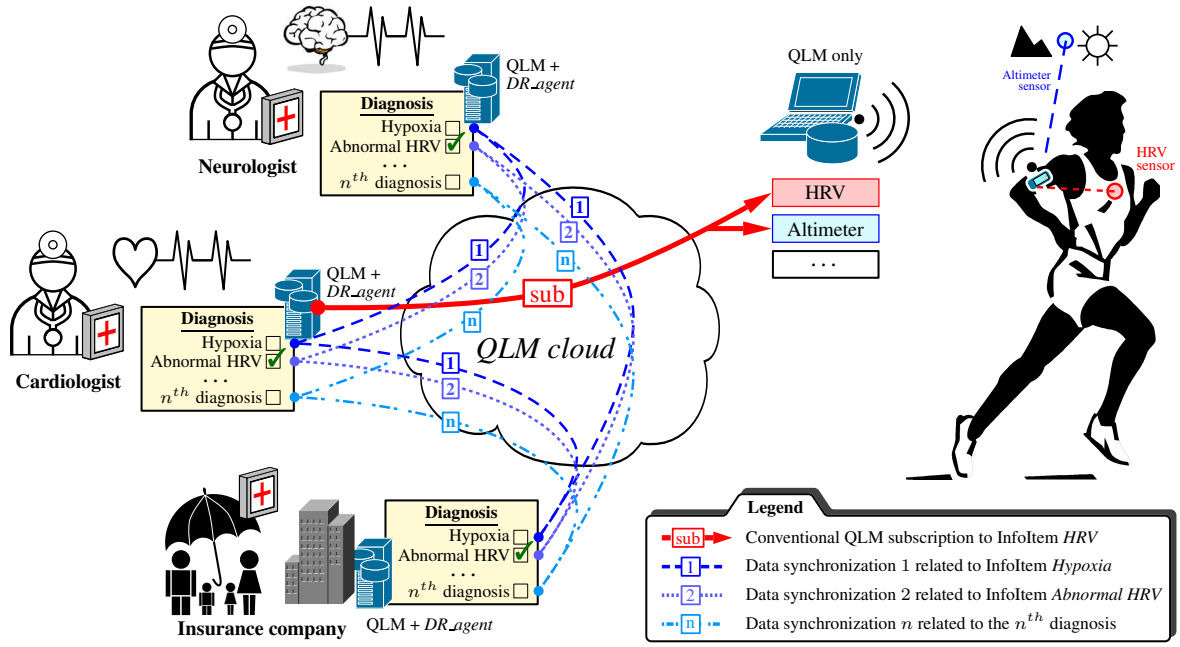


Figure 14: Healthcare environment and data synchronization between particular healthcare providers and companies

to authorized persons/organizations. The information system of each actor is augmented with the QLM messaging capabilities, as emphasized in Figure 14.

The cardiologist would like to subscribe to InfoItems *HRV* and *Altimeter,* which are located in the jogger's database. Therefore, each time the jogger goes home and information is updated in the database, it will automatically be pushed to the cardiologist's database. Because only one actor (the cardiologist) is interested in receiving such information, it is possible to directly use the subscription mechanism defined in QLM-MI (i.e., there is no need to implement a data synchronization via the *DR_agent*). Given this observation, the cardiologist sends the QLM subscription request detailed in Figure 15 to the jogger database to subscribe to InfoItems *HRV* and *Altimeter* (see rows 7 and 8 of the XML message) by including his own address as the *callback* (see row 2) and by setting the interval parameter to "-1" (see row 2) that indicates that the cardiologist requests an *event-based* subscription. Let us note that the "id" of the Object contained by the message (see row 6 in Figure 15) corresponds to the jogger's INSEE number, which is globally unique. Once new values are published in the jogger's database and then pushed to the cardiologist's database, it is thus possible to analyze and diagnose any disorder or disease (e.g., an abnormal HRV). The cardiologist could even further subscribe (in "real-time") to additional information in the jogger's database. To avoid making the scenario too complex, we assume that the cardiologist only formulates diagnoses based on the HRV and Altimeter values. In this regard, both the neurologist and the insurance company are interested in receiving such diagnoses (see the diagnosis list example given in Figure 15 considering n diagnoses). Notifying the neurologist about an abnormal HRV could, for instance, enable this expert to predict future epilepsy seizures, as studied in [61, 62] (effects of epilepsy on the HRV in the pre-ictal phase are analyzed), and eventually allow him to complement or modify the current diagnoses list (according to his access rights).

```
1  <qlmEnvelope xmlns="QLMmi.xsd" version="1.0"
       ttl="-1">
2    <read msgformat="QLMdf.xsd" interval="-1"
         callback="http://207.46.130.1//Cardio_DB
         ">
3      <msg>
4        <Objects xmlns="QLMdf.xsd">
5          <Object type="HRV">
6            <id>186055763003082</id>
7            <InfoItem name="HRV"/>
8            <InfoItem name="Altimeter"/>
9          </Object>
10       </Objects>
11     </msg>
12   </read>
13 </qlmEnvelope>
```

Figure 15: QLM subscription (i.e., read) request sent by the cardiologist to the jogger's database to subscribe InfoItem HRV

Because several health service providers are allowed to access and modify some of the diagnoses contained in this list, data synchronization strategies are required among these actors, depending on who is allowed to modify them. If a diagnosis can be modified by several health service providers, a synchronization of type A-U_e may be more appropriate. These different (or potential) diagnosis synchronizations are

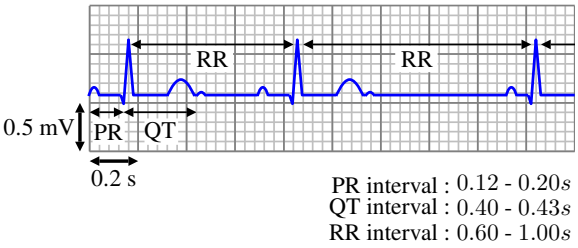


Figure 16: HRV also known as RR intervals

represented in Figure 14 through lines denoted by 1, 2,..., n. More concretely, InfoItems that can be synchronized in this platform are the different diagnoses contained in the "diagnosis list". Section 6.3 focuses on data synchronization 2, i.e., the synchronization of the diagnosis/InfoItem named *Abnormal_HRV* that can take two values: `YES` or `NO`.

6.3. Data synchronization 2

The HRV indicates the variation of beat to beat intervals, also known as RR intervals. Figure 16 gives insight into an HRV signal and related intervals. RR intervals usually range between 0.6 and 1.0 *s* [63]. In our case, both the cardiologist and neurologist can diagnose an abnormal HRV[8] and, consequently, we decide to implement a synchronization of type A-U_e between these actors. This implies that the *DS_agent* is required in the information system of each of these actors (not in the jogger system), as emphasized in Figure 14.

Figure 17 presents a scenario relying on data synchronization 2, which is presented in the form of a sequence diagram. As previously described, two main phases for synchronizing an InfoItem are required:

i. the initialization of data synchronization among the actors;

ii. the ongoing synchronization process when further events and information are entered in the system (e.g., information uploaded in the jogger's database, detection of abnormal HRV...).

These two phases are highlighted in Figure 17; however, this section only focuses on phase *ii.* (see the previous case study for more information about phase *i.*). When considering phase *ii.*, data synchronization 2 (related to diagnosis *Abnormal_HRV*) is ready for use among the three health providers/companies, and the cardiologist is subscribed to InfoItem *HRV* located in the jogger's database.

The jogger goes home after jogging, and data are automatically uploaded into his database (see "jogging 1" in Figure 17). Because the cardiologist subscribed to InfoItem *HRV*, related values are therefore pushed to the appropriate system (see communication denoted by "1" in Figure 17). Based on these values, an abnormal HRV is detected (values too often exceed 1.0 *s*), and accordingly, the diagnosis/InfoItem *Abnormal_HRV* is changed from `NO` to `YES` (see Figure 17). The modification of this InfoItem corresponds to a write request in the cardiologist system, and because it is synchronized using the "A-U_e" technique, box connectors 1, 3, 4, 7, 11, 12, 13, and 2 from the *handle()* function in Figure 7 are solicited, leading to the sending of two QLM responses to the other systems involved in data synchronization 2 (see arrows denoted by "2a" and "2b" in Figure 17).

A pre-defined service in the neurologist system has been configured to notify the neurologist when the diagnosis list is updated[9]. This corresponds to the frame named "Alarm()" in Figure 17. Based on this notification, the neurologist would like to obtain the exact HRV values to study possible epilepsy syndromes. To comply with this requirement, the RESTful QLM "discovery" mechanism proposed in the QLM standards is used to discover what types of jogger health information the cardiologist has access to. Figure 18 gives insight into how such a mechanism can be used via the Unix `wget` utility. This request consists, in this example, of a URL where it is possible to retrieve all QLM "Object"(s) and their related InfoItems and other Object sub-elements that are currently available in the cardiologist system, and thus accessible to the neurologist. In our study, the jogger's INSEE number is known by all doctors and is used to access the "Objects" containing jogger information. `wget_1` given in Figure 18 shows how this request is performed using the jogger's INSEE number (French National Institute for Statistics and Economic Studies) and what information is returned in the response (i.e., information that is accessible), namely, InfoItems *HRV* and *Altimeter*. The `wget_1` command and the related response correspond to arrows denoted by "3" and "4" in Figure 14. The neurologist therefore decides to request the retrieval of historical HRV values from the cardiologist's database (such values could even be subscribed to and received as future values). Such a request and its response respectively correspond to the arrows denoted by "5" and "6" in Figure 17. The retrieved values are then analyzed, leading to the conclusion that the jogger has a high risk of having an epilepsy seizure. The diagnosis list is then updated in the neurologist system from `NO` to `YES`, as highlighted in Figure 17. However, in our platform,

[8] The neurologist can diagnose such a disorder by performing, for instance, tests directly on the patient.

[9] A QLM write in our study is used to send the notification to a mobile phone.

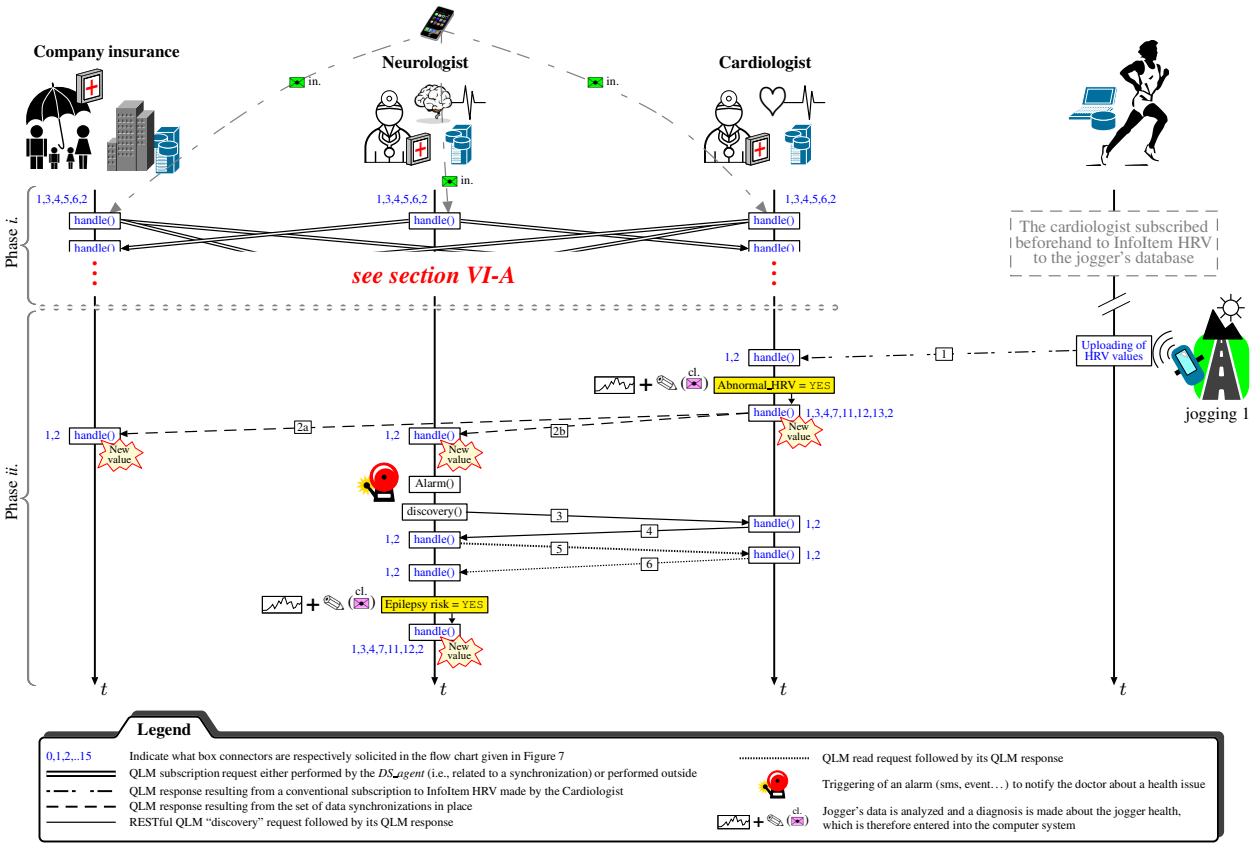


Figure 17: Sequence diagram related to the initialization and ongoing process of data synchronization 3 ("A-U_e")

no synchronization regarding the InfoItem/diagnosis named *Epilepsy_risk* was set up between the three actors. As a consequence, the new diagnosis value is not propagated to other healthcare providers/companies because InfoItem *Epilepsy_risk* is not registered in $\mathcal{R}$, as highlighted in Figure 17 (box connectors 1, 3, 4, 7, 11, 12, and 2 from the flow chart in Figure 7 are solicited). It is, nonetheless, simple to further activate such a synchronization (if necessary), as was done previously for the diagnosis *Abnormal_HRV*.

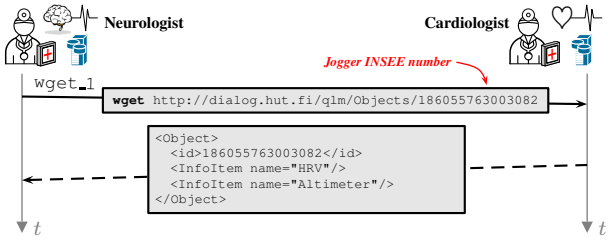


Figure 18: RESTful QLM "discovery" mechanism used by doctors

6.4. Synthesis

The case study carried out on home automation placed major emphasis on the *initiating* phase of our data synchronization proposals and on showing that the synchronization/replication constraints were respected. In the healthcare scenario, we showed how these synchronization strategies can take advantage of other generic interfaces defined in QLM (conventional subscription, QLM "discovery" mechanism, *etc.*). It should be noted that in both case studies, we did not consider any domain-specific application such as home automation software (e.g., OpenHAB®) or healthcare software (e.g., McKESSON®), which require further developments and frameworks to be integrated into the IoT, or to be more exact, into the QLM cloud for making it possible to use the proposed multi-*DS_agent* system. First investigations on such an integration (regardless of the data synchronization aspect) have nonetheless been carried out in [64].

The support of traditional data synchronization mechanisms based upon QLM now lays a solid groundwork to develop and propose more advanced (PLM) services to users and professionals, as has been shown through a few examples (alarm triggers, data analysis, discovery of information and services). In this regard, new strategies for context-aware data distribution could further be developed to automatically set up the appropriate data synchronization between relevant product stakeholders and systems. The recent survey on context aware data distribution in the IoT field carried out by Bellavista et al. [65] provides

a threefold classification of distribution-related services (including synchronization services) as: *i) unaware* (the service level neither reaches nor influences run-time adaptation support strategies); *ii) partially-aware* (services can influence the run-time adaptation support); or *iii) totally-aware* (the run-time adaptation support does not perform anything on its own, and it is the service level that completely drives reconfigurations). Following this classification, it can be stated that our multi-*DS_agent* system is currently unaware of the product context and environment. Indeed, currently, the parameters specified in the `initiate()` or `remove()` function are entered by the user/engineer (see Figure 12). However, the standardized and advanced QLM interfaces offer new possibilities to design *partially-aware* or *totally-aware* algorithms/services to automatically set the appropriate parameter values according to the product context, the user needs, and possibly other relevant surrounding information, thus making the synchronization setting transparent to the user/engineer. Such a new generation of context-aware services is of particular importance in the context of PLM because product stakeholders are increasingly looking for full-services that make it possible to retrieve consistent information about their products under in-use conditions, to learn how their products behave, and to self-react according to the product and user contexts. This is all the more important in light of context-aware distribution & synchronization services, as reported by Bellavista et al. who state that such services able to self-adapt autonomously depending on current management conditions is still an unexplored research field.

On a final note, in both scenarios and especially in healthcare environments, we deliberately neglected issues of user's privacy and security policies because the primary goal was to validate the feasibility of the multi-*DS_agent* system.

7. Conclusion and discussion

Networks, systems, and products of the 21st century transcend the traditional organizational boundaries because every "thing" will literally become "connected" to the so-called IoT. To a certain extent, the IoT concept also relies on the automatic capture of observations of physical objects at various locations and times, their movements between locations, sensor data collected from sensors attached to the objects, and so forth. This vision of the IoT is closely linked to the concept of Product Lifecycle Management (PLM), which is a strategic approach to enable all participants and decision-makers to have a clear, shared understanding of the product life. It is a fact today that more advanced applications, services, and interfaces must be built in the IoT to enable interactions among various "things" throughout the product lifecycle (devices, products, users, enterprise information systems…). In the past few years, there has been growing interest in proposing standards that fulfill these requirements, but so far, none of them have become the official or *de facto* IoT standard. Recently, Quantum Lifecycle Management (QLM) messaging specifications have been developed and proposed to become such a standard (official QLM specifications should be made public during 2014). QLM standards provide high-level abstraction communication interfaces that significantly enhance data exchange interoperability in the IoT or, from another perspective, in PLM environments.

Our research claims that proposed solutions, models, and tools sufficiently portable and independent of specific vendors, networks, or applications have to be developed based on such generic interfaces, to the extent possible. This paper focuses on an essential aspect of PLM, which involves improving "information consistency" throughout the product lifetime using such independent and interoperable solutions. Information consistency can be maintained by implementing, among others, data synchronization mechanisms. Accordingly, this paper investigates and develops data synchronization models based upon QLM that facilitate the synchronization of any product-related information among various and distinct lifecycle entities (e.g., distinct organizations, networks, information systems, devices…). The originality of this research initiative lies in the fact that these models do not require the addition of any software or hardware with QLM, which significantly increases the scope of possible applications. As a first step, this paper focuses on "asynchronous" data synchronization models. A software agent, referred to as a *DS_agent*, has been developed to enable the integration of the conceptual models investigated in this paper. It is very important to note that these models are generic enough to be used with any other Messaging protocol (i.e., other than QLM) under the condition that this protocol must support, at least, the following functionalities: *i)* support for *read* and *write* operations, *ii)* support for an *event-based* subscription mechanism with *callback address*, and *iii)* support for *asynchronous communications* (e.g., using a TTL parameter). Therefore, if QLM fails in its attempt to become the *de facto* standard for IoT data exchange, the conceptual models proposed in this paper could be easily integrated into other standards/protocols.

The *DS_agent* has been implemented in various platforms that are presented in this paper, namely, home automation and healthcare platforms, which allowed us to demonstrate its practical integration with QLM. These scenarios also provide first examples on how our proposals can help with defining and offering new types of services to customers and professionals. If QLM standards could come to be used in numerous

organizations and applications from different lifecycle phases, the support of traditional data synchronization principles based upon QLM could lead to substantial time and cost savings.

References

[1] K. Ashton, Internet things – MIT, embedded technology and the next internet revolution, in: Baltic Conventions, The Commonwealth Conference and Events Centre, London, (2000).

[2] N. Gershenfeld, R. Krikorian, D. Cohen, The Internet of Things, Scientific American 291(4) (2004) 76–81.

[3] L. Atzori, A. Iera, G. Morabito, The Internet of Things: A survey, Journal 54(15) (2010) 2787–2805.

[4] M. Harrison, The 'Internet of Things' and commerce, XRDS: Crossroads, The ACM Magazine for Students 17(3) (2011) 19–22.

[5] A. Sääksvuori, A. Immonen, Product lifecycle management, Springer, 2002.

[6] J. Stark, Product lifecycle management: 21st century paradigm for product realisation, Springer, 2011.

[7] F. Ameri, D. Dutta, Product lifecycle management: closing the knowledge loops, Computer-Aided Design and Applications 2(5) (2005) 577–590.

[8] G. Chryssolouris, D. Mavrikios, N. Papakostas, D. Mourtzis, G. Michalos, K. Georgoulias, Digital manufacturing: history perspectives, and outlook, in: Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 223(5) (2009), 451–462.

[9] G. Meyer, K. Främling, J. Holmström, Intelligent products: A survey, Computers in Industry 60(3) (2009) 137–148.

[10] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos, Context Aware Computing for The Internet of Things: A Survey, IEEE Communications surveys & Tutorials (99) (2013) 1–41.

[11] K. Främling, J. Holmström, J. Loukkola, J. Nyman, A. Kaustell, Sustainable PLM through Intelligent Products, Engineering Applications of Artificial Intelligence 26(2) (2013) 789–799.

[12] D. Chen, G. Doumeingts, F. Vernadat, Architectures for enterprise integration and interoperability: Past, present and future, Computers in Industry 59(7) (2008) 647–659.

[13] M. Baldauf, S. Dustdar, F. Rosenberg, A survey on context-aware systems, International Journal of Ad Hoc and Ubiquitous Computing 2(4) (2007) 263–277.

[14] D. Uckelmann, M. Harrison, F. Michahelles, An architectural approach towards the future Internet of Things, in: Springer-Verlag Berlin Heidelberg (2011), 1–24.

[15] K. Främling, S. Kubler, A. Buda, Universal messaging standards for the IoT from a lifecycle management perspective, IEEE Internet of Things Journal (2014), DOI: 10.1109/JIOT.2014.2332005.

[16] M. Harrison, A. K. Parlikad, Lifecycle ID and lifecycle data management, Tech. Rep.: AUTO-ID Labs, AEROID-CAM-005, 2006.

[17] A. H. Huang, A model for environmentally sustainable information systems development, Journal of Computer Information Systems 49(4) (2009) 114–121.

[18] M. Butrico, N. Cohen, J. Givler, A. Mohindra, A. Purakayastha, D. G. Shea, J. Cheng, D. Clare, G. Fisher, R. Scott, Y. Sun, M. Wone, Q. Zondervan, Enterprise data access from mobile computers: an end-to-end story, in: 10th International Workshop on Research Issues in Data Engineering, (2000), 9–16.

[19] K.-D. Thoben, J. Eschenbächer, H. Jagdev, Extended products: evolving traditional product concepts, in: 7th International conference on concurrent enterprising engineering the knowledge economy through co-operation, Bremen, (2001), 429–439.

[20] K. Främling, T. Ala-Risku, M. Kärkkäinen, J. Holmström, Agent-based model for managing composite product information, Computers in Industry 57(1) (2006) 72–81.

[21] M. Kärkkäinen, T. Ala-Risku, K. Främling, The product centric approach: a solution to supply network information management problems?, Computers in Industry 52(2) (2003) 147–159.

[22] D. McFarlane, S. Sarma, J. L. Chirn, C. Y. Wong, K. Ashton, The intelligent product in manufacturing control and management, Journal of EAIA, (2002) 54–64.

[23] M. Kärkkäinen, J. Holmström, K. Främling, K. Artto, Intelligent products–a step towards a more effective project delivery chain, Computers in Industry 50(2) (2013) 141–151.

[24] Y. Sallez, T. Berger, D. Deneux, D. Trentesaux, The lifecycle of active and intelligent products: The augmentation concept, International Journal of Computer Integrated Manufacturing 23(10) (2010) 905–924.

[25] D. Kiritsis, A. Bufardi, P. Xirouchakis, Research issues on Product Lifecycle Management and information tracking using smart embedded systems, Advanced Engineering Informatics 17(3) (2003) 189–202.

[26] D. Kiritsis, Closed-loop PLM for intelligent products in the era of the Internet of Things, Computer-Aided Design 43(5) (2011) 479–501.

[27] L. N. Van Wassenhove, V. D. R. Guide, Closed-loop supply chains, Pittsburgh, 2003.

[28] D. S. Rogers, R. S. Tibben-Lembke, Going backwards: reverse logistics trends and practices, Reverse Logistics Executive Council Pittsburgh, PA, 1999.

[29] S. Dynes, L. Kolbe, R. Schierholz, Information Security in the Extended Enterprise: a research agenda, in: 13th Americas conference on information systems, 4322–4333, 2007.

[30] A. Koronios, D. Nastasie, V. Chanana, A. Haider, Integration through standards–An overview of international standards for engineering asset management, in: 4th International Conference on Condition Monitoring, Harrogate, (2007), 11–14.

[31] H. Jun, D. Kiritsis, P. Xirouchakis, Research issues on closed-loop PLM, Computers in Industry 58(8) (2007) 855–868.

[32] K. Michael, L. McCathie, The pros and cons of RFID in supply chain management, in: International conference on mobile business, (2005), 623–629.

[33] G. C. Initiative, Capgemini, Global data synchronisation at work in the real world-Illustrating the business benefits, Tech. Rep.: Global Commerce Initiative and Capgemini, 2005.

[34] M. Tajima, Strategic value of RFID in supply chain management, Journal of purchasing and supply management 13(4) (2007) 261–273.

[35] S. Lockhead, The Global Data Synchronisation Network (GDSN): Technology and standards improving supply chain efficiency, in: IEEE International Technology Management Conference, (2011), 630–637.

[36] S. Suzuki, M. Harrison, Data synchronization specification. Aerospace-ID program report., Tech. Rep., Auto-ID Labs, University of Cambridge, 2006.

[37] F. Thiesse, C. Floerkemeier, M. Harrison, F. Michahelles, C. Roduner, Technology, standards, and real-world deployments of the EPC network, IEEE Internet Computing 13(2) (2009) 36–43.

[38] K. Nakatani, T.-T. Chuang, D. Zhou, Data synchronization technology: standards, business values and implications, Communications of the Association for Information Systems 17(1) (2006) 962–994.

[39] Pumatech, Invasion of the data snatchers, in: White paper, 1999.

[40] S. Agarwal, D. Starobinski, A. Trachtenberg, On the scalability of data synchronization protocols for PDAs and mobile devices, IEEE Network 16(4) (2002) 22–28.

[41] R. M. Mettala, A. Purakayastha, P. Thompson, SyncML: Synchronizing and managing your mobile data, Prentice Hall Professional, 2002.

[42] J. Pak, K. Park, UbiMMS: an ubiquitous medication moni-

toring system based on remote device management methods, The Journal of Healthcare Information Management 41(1) (2012) 26–30.

[43] Y.-C. Chen, H.-C. C., M.-D. Tsai, H. Chang, C.-F. Chong, Development of a personal digital assistant-based wireless application in clinical practice, Computer methods and programs in biomedicine 85(2) (2007) 181–184.

[44] F. Gil-Castineira, D. Chaves-Dieguez, F. González-Castaño, Integration of nomadic devices with automotive user interfaces, IEEE Transactions on Consumer Electronics 55(1) (2009) 34–41.

[45] G. Lu, D. Seed, M. Starsinic, C. Wang, P. Russell, Enabling, Smart Grid with ETSI M2M Standards, in: IEEE Wireless Communications and Networking Conference Workshops, (2012), 148–153.

[46] M. Castro, A. J. Jara, A. F. G. Skarmeta, Smart Lighting solutions for Smart Cities, in: 27th International Conference on Advanced Information Networking and Applications Workshops, (2013), 1374–1379.

[47] Y. Tian, J.-P. Li, Research and implementation of data synchronization with SyncML, in: IEEE International Conference on Wavelet Active Media Technology and Information Processing, (2012), 302–304.

[48] S. Kubler, M. Madhikermi, A. Buda, K. Främling, 10th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Tokyo, (2012).

[49] R. Ladin, B. Liskov, L. Shrira, S. Ghemawat, Providing high availability using lazy replication, ACM Transactions on Computer Systems 10(4) (1992) 391–425.

[50] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, Database replication techniques: A three parameter classification, in: 19th IEEE Symposium on Reliable Distributed Systems, (2000), 206–215.

[51] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, Understanding replication in databases and distributed systems, in: ICDCS, (2000), 464–487.

[52] J. Gray, P. Helland, P. O'Neil, D. Shasha, The dangers of replication and a solution, SIGMOD Rec. 25 (1996) 173–182.

[53] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, A. Demers, Flexible update propagation for weakly consistent replication, in: Operating Systems Review, SIGOPS, (1997), 288–301.

[54] A. T. M. Aerts, N. B. Szirbik, J. B. M Goossenaerts, A flexible, agent-based ICT architecture for virtual enterprises, Computers in Industry 49(3) (2002) 311–327.

[55] P. Valckenaers, M. Kollingbaum, H. Van Brussel, Multi-agent coordination and control using stigmergy, Computers in Industry 53(1) (2004) 75–96.

[56] L. Monostori, J. Váncza, S. R. T. Kumara, Agent-based systems for manufacturing, CIRP Annals-Manufacturing Technology 55(2) (2006) 697–720.

[57] W. Shen, D. H. Norrie, Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey, Knowledge and Information Systems, an International Journal 1(1999) 129–156.

[58] J. Ferber, Multi-agent systems: an introduction to distributed artificial intelligence, Addison-Wesley Reading, 1999.

[59] J. Leber, Health Insurer's App Helps Users Track Themselves, http://www.technologyreview.com/news/516176/health-insurers-app-helps-users-track-themselves, 2013.

[60] K. Reijonsaari, A. Vehtari, W. Van Mechelen, T. Aro, S. Taimela, The effectiveness of physical activity monitoring and distance counselling in an occupational health setting – a research protocol for a randomised controlled trial (CoAct), BMC public health 9(1) (2009) 494.

[61] D. H. Kerem, A. B. Geva, Forecasting epilepsy from the heart rate signal, Medical and Biological Engineering and Computing 43(2) (2005) 230–239.

[62] S. Behbahani, N. J. Dabanloo, A. M. Nasrabadi, G. Attarodi, C. A. Teixeira, A. Dourado, Epileptic seizure behaviour from the perspective of heart rate variability, in: Computing in Cardiology, Krakow, (2012), 117–120.

[63] E. Rodriguez, A. de Luca, M. Meraz, J. Alvarez-Ramirez, Breakdown of scaling properties in abnormal heart rate variability, Journal of Applied Research and Technology 4(1) (2006) 82–97.

[64] N. Shrestha, S. Kubler, K. Främling, Standardized framework for integrating domain-specific applications into the IoT, in: 2nd International Conference on Future Internet of Things and Cloud, (2014).

[65] P. Bellavista, A. Corradi, M. Fanelli, L. Foschini, A Survey of context data distribution for mobile ubiquitous systems, ACM Computing Surveys 45(1) (2013) 1–49.