P2P Data Synchronization for Product Lifecycle Management

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

William Derigent
Université de Lorraine, CRAN, UMR 7039, Vandœuvre-lès-Nancy F-54506, France
Email: firstname.lastname@univ-lorraine.fr

Abstract—Intelligent products are the key driver for the so-called Internet of Things (IoT). To facilitate the IoT, Product Lifecycle Management (PLM) requires the automatic capture of events related to physical objects at various locations and times. Nonetheless, more advanced and standardized communication interfaces must be designed to facilitate interactions between various types of “things” throughout the product lifecycle (devices, products, users…). Our research claims that such standardized interfaces are a necessary condition to develop solutions, models, and tools that are sufficiently portable and independent of specific vendors and operating systems. Based on this observation, this paper describes strategies to improve “information sustainability” in PLM systems using standardized communication interfaces defined by a recent standard proposal for the IoT, called Quantum Lifecycle Management (QLM). More concretely, data synchronization models based upon QLM standards are developed to enable the synchronization of product-related information among various lifecycle systems, networks, and organizations. Our proposals are implemented and assessed based on two distinct platforms defined in the healthcare and home automation sectors.

Index Terms—Product Lifecycle Management – Internet of Things – Data synchronization – Multi-system agents – Quantum Lifecycle Management

I. INTRODUCTION

In the so-called “Internet of Things” (IoT), mobile users and objects will be able to dynamically discover and interact in an impromptu manner with heterogeneous computing platforms, physical resources, and virtual data and environments [1], [2]. To a certain extent, the IoT vision also 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 [3], [4], [5]. This concept is closely linked to the concept of Product Lifecycle Management (PLM) [6], [7], 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. In traditional product lifecycles, the advent of the IoT has radically reduced the communication cost for remote collaboration and coordination in PLM projects, which leads to transcending the traditional organizational boundaries and space limitations [8], [9].

These new boundaries reveal a real need for new types of interactions between various types of “things” in the IoT (devices, computers, users, and information systems in general) to help companies to deal with complex, changing product environments and to meet the new organizational and customer needs [10], [11]. Despite this observation, no proper agreement on a common standard for IoT data exchange between organizations, whether in terms of data structure or data communication, has yet been reached or even proposed [12], [13]. Quantum Lifecycle Management (QLM) messaging standards1 have been developed and are proposed as a standard that would fulfill the main IoT requirements [10]. The “QLM cloud” (see Figure 2) aims to provide high-level abstraction interfaces to enable the kind of information exchange required in the IoT, no matter the technology and the system infrastructure, thus contributing to the improvement of interoperability among all product stakeholders [14]. As stated in [15], [16], [17], such standardized interfaces are a necessary condition to develop solutions, models, and tools sufficiently portable and independent of specific vendors, applications, and operating systems. Based on this observation, this paper presents strategies to improve “information sustainabil-

1http://www.opengroup.org/qlm/
ity” in PLM systems using the generic communication interfaces defined by the QLM standards. The main characteristic that makes 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) [18], [19]. Information consistency can be maintained by implementing data synchronization mechanisms [20]. As a result, this paper investigates and develops data synchronization models based upon QLM that make it possible to synchronize product-related information among various and distinct lifecycle actors (e.g., distinct organizations, networks, information systems, devices...), whose originality lies in the fact that: i) the data synchronization models developed in this paper do not require the addition of any software or hardware with QLM that significantly increases the scope of possible applications, and ii) such models could be used in conjunction with any other messaging protocol that provides the set of interfaces needed.

Section II provides both the IoT and data synchronization backgrounds from a PLM perspective. Section III provides insight into traditional data synchronization principles that ought to be integrated using the generic QLM interfaces, which are introduced in section IV. Section V presents the generic data synchronization models that are developed in this paper based on the QLM interfaces. These models are then implemented and validated by considering two distinct platforms in the healthcare sector and home automation; the conclusions follow.

II. 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 II-A 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 II-B.

A. 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 [6], [7]. 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 [21]. Figure 1 provides insight into a traditional product lifecycle from the Beginning of Life (BoL), including the design, production and distribution of the product, through the 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” [22] 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” [23], “product-centric” PLM [24], and “intelligent products” [25], [26] 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. This project introduced the closed-loop PLM® paradigm [27], [28], recently renamed CL2M (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, CL2M and similar paradigms such as the “Closed-Loop Supply Chains” [29], [30] or “Reverse Logistics” [31] 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 organiz-
2) information manageability: to efficiently and properly process large amounts of raw data [13];
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 [33], [34];
4) information visibility: to make data available for any system, anywhere, and at anytime. The CL²M consortium³ defines the visibility of the information as the possibility to gather, process, and exchange the desired information throughout the whole life of an entity [35], [36];
5) information sustainability: to make data capable of outliving systems, while being consistent [14].

These five aspects must be properly addressed for an efficient PLM, as highlighted in Figure 1. 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) [7]. 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 Group⁴. QLM messaging standards are derived from the PMI and are intended to provide sufficiently generic and generally applicable application-level IoT messaging standards to fulfill the main IoT requirements [10]. 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.

B. 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 [37], [18], [38]. 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 [39]. 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

³http://www.cl2m.com
⁴http://www.opengroup.org/qlm/
Global Data Synchronization Network\(^5\) (GDSN), and the Auto-ID centers (founders of EPC information services that became part of GS1) \(^{[40]}\), \(^{[41]}\). 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 \(^{[42]}\). 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 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) \(^{[14]}\).

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 \(^{[43]}\), and CPIISync \(^{[44]}\). 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\(^6\) (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 \(^{[45]}\). 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 \(^{[46]}\), \(^{[47]}\), automotive applications \(^{[48]}\), or smart grid and cities \(^{[49]}\), \(^{[50]}\). 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 \(^{[51]}\), the synchronization model defined in the specifications is closer to a client-server relationship established between one or more mobile devices and a server than a real peer-to-peer situation \(^{[45]}\). 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 messaging 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 and \(ii\) 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); see \(^{[10]}\) 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. Section III introduces the main techniques from the literature that are used to synchronize information between different systems.

### III. 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 \(^{[52]}\);
- **Where do the updates have to be performed?** Two principles exist: Primary copy (P\(_c\)), which allows

\(^5\)GDSN is associated to GS1 Data Standards: www.gs1.org

\(^6\)http://openmobilealliance.org
one and only one node to modify the data, and Update everywhere (Uₜ), 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[53]: (i) A-Uₜ, (ii) A-Pₜ, (iii) S-Uₜ, and (iv) S-Pₜ. Each technique has pros and cons, as detailed in [54]. 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 [55] (see Figure 2). 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, we investigate the feasibility of implementing the two asynchronous techniques (i.e., A-Uₜ and A-Pₜ) using the standardized QLM interfaces. An investigation of synchronous techniques will be conducted in the future.

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-Uₜ 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-Uₜ 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 [56], [57]. This problem is avoided when using the A-Pₜ technique (see Figure 3(b)) because only the node designated as Pₜ is allowed to modify k. Let node C be designated as Pₜ 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 Pₜ 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 IV.

IV. QLM MESSAGING

QLM messaging specifications consist of two standards proposals [10]: 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...
Fig. 4. QLM Data Format: generic “Object” tree and example of a QLM message relying on that tree

`<qlm:Envelope xmlns="QLMmi.xsd" version="1.0" tll="0.0">`
`<qlm:read msgformat="QLMdf.xsd">`
`<Objects xmlns="QLMdf.xsd">`
`<Object type="Fridge">`
`<id>Smartfridge233410c</id>`
`<InfoItem name="Door_status"></InfoItem>`
`<InfoItem name="Thermostat"></InfoItem>`
`</Object>`
`</Objects>`
`</qlm:read>`
`</qlm:Envelope>`

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. 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.

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. 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 value is specified in row 1 and the operation type is specified in row 2 (“read”). Other relevant interfaces are presented in [10] like the “publication and discovery” of new data sources, services and meta-data by using “RESTful” URL-based queries (including discovery by search engines).

Section IV 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.

V. 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” [23]. 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 [58]. An agent may operate in environments where other agents operate, with the whole operating as a multi-agent system [59], [60].

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 V-A and V-B.

A. “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.

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 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

\begin{algorithm}[H]
\caption{Algorithm 1: initiate($II_n, L_n, tech$)}
\begin{algorithmic}[1]
\Statex \hspace{1em} \textbf{output: } $M_{in}$
\Statex \hspace{1em} \begin{align*}
\quad M_{in} \leftarrow 0; & \quad // \text{Initialize the set of messages to be sent to empty }
\quad obj \leftarrow DSinit; & \quad // \text{Set the type of the } *\text{Object}\text{* element of the message to } *\text{DSinit*}
\end{align*}
\Statex \hspace{1em} \textbf{forall} the $@j \in L_n$ do // for all nodes $\in L_n$
\Statex \hspace{2em} $M = \text{qlmWrite}(obj, II_n, L_n, tech, @j);$ // Create the appropriate QLM write message according to the specified inputs
\Statex \hspace{2em} $M_{in} \leftarrow M_{in} \cup M$; // Add message to $M_{in}$
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[H]
\caption{Algorithm 2: remove($II_n, L_n, tech$)}
\begin{algorithmic}[1]
\Statex \hspace{1em} \textbf{output: } $M_{re}$
\Statex \hspace{1em} \begin{align*}
\quad M_{re} \leftarrow 0; & \quad // \text{Initialize the set of messages to be sent to empty }
\quad obj \leftarrow DSrem; & \quad // \text{Set the type of the } *\text{Object}\text{* element of the message to } *\text{DSrem*}
\end{align*}
\Statex \hspace{1em} \textbf{forall} the $@j \in L_n$ do // for all nodes $\in L_n$
\Statex \hspace{2em} $M = \text{qlmWrite}(obj, II_n, L_n, tech, @j);$ // Create the appropriate QLM write message according to the specified inputs
\Statex \hspace{2em} $M_{re} \leftarrow M_{re} \cup M$;
\end{algorithmic}
\end{algorithm}
For instance, considering node B in cell c generates three QLM messages of type “IIn” (see rows 8, 11, and 14-16). The process is similar for removing a data synchronization in the system. 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:

- \( I_{I_\text{A}} \) = Temperature1326;
- \( tech \) = A-Pc with node C designated as Pc;
- \( L_n \) = \{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 Pc that consists of positioning the Pc node address at the first position in \( L_n \).

Using Algorithm 1, the \( DS_{\text{agent}} \) 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 subelement “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.

The set of QLM nodes ($L_n$) must then start to implement (or remove) the data synchronization in the whole multi-$\text{DS}_\text{agent}$ 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 $\text{handle()}$ 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 "\text{QLM}(\text{in})" (see cell $c$), of type "\text{QLM}(\text{re})" (see cell $e$), or of any other type, i.e. any message that contains “Object” types different from $\text{DSinit}$ or $\text{DSrem}$ (corresponds to cell $d$). Unlike $\text{initiate()}$ and $\text{remove()}$ options, $\text{handle()}$ is referred to as message-based reception because it all starts with the reception of a QLM message, as shown in Figure 5.

B. “handle” synchronization

This option is considered the core of the $\text{DS}_\text{agent}$ because it is in charge of creating synergy in the multi-$\text{DS}_\text{agent}$ system for initiating, handling, and even removing a data synchronization according to the incoming messages. More concretely, the $\text{handle()}$ option is in charge of:

i) creating the appropriate subscriptions in the multi-$\text{DS}_\text{agent}$ 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-$\text{DS}_\text{agent}$ 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 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 $\text{DSinit}$ (see box connector “4”), $\text{DSrem}$ (see box connector “7”), or of other types (see box connectors “11”). Sections V-B1, V-B2, and V-B3 respectively detail the actions to be undertaken by the $\text{DS}_\text{agent}$ for each of these object types.

1) Object – $\text{DSinit}$: When a node receives a message $M$ containing an “Object” of type $\text{DSinit}$, 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 $U_c$).
Because the $U_n$ principle allows all nodes carrying a replica to modify it, $z(z-1)$ subscriptions must be originally defined in the system with $z$ being 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 $I_{II_n}$. Three situations could potentially occur: a client could request to modify $I_{II_n}$ 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 $I_{II_n}$ on node C (see arrows denoted by $\circ$ in Figure 8(a)) by defining an event-based subscription and by providing their respective address as a callback. Hence, each time $I_{II_n}$ is modified on node C (see arrow denoted by $\circ$), all subscribers receive a notification with its new value (see arrows denoted by $\circ$). 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) ($B \rightarrow C; D \rightarrow C; C \rightarrow B; D \rightarrow B; B \rightarrow D$ and $C \rightarrow D$). 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 Infotlem 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 $I_{II_n}$ 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 $I_{II_n}$, 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-$U_n$, is computed using the function named sub() in box connector “6” in Figure 7, which is detailed in Algorithm 3 (see comments provided in this algorithm for more details).

2) Object – $DSrem$: When a $DS_{agent}$ receives a message $M$ with an “Object” of type $DSrem$, the set of Infotlems that compose this object (i.e., $I_{II_n}$, tech, and $L_n$) are first retrieved (see box connector denoted by “8” in the flow chart). Based on these Infotlem values, it is then necessary to cancel the set of subscriptions that have been set up among the QLM nodes beforehand (regarding $I_{II_n}$). 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 $\text{sub}()$ (i.e., Algorithm 3), except that the generated messages are QLM cancel requests and not QLM subscription (read) requests.

3) Object – other types: When a $\text{DS}_\text{agent}$ receives a Write message $M$ with an Object type different than $\text{DSInit}$ and $\text{DSrem}$, 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 $\mathcal{R}$ (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 “13” and “2”). Otherwise, it is necessary to check whether the current node (i.e., $\@\text{cur}$) 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 using QLM standards; otherwise, $\mathcal{I}(n)$ has to be forwarded to the $P_c$ node, which is achieved thanks to the function named $\text{forward()}$ in box connector “15”.

VI. CASE STUDY

Two platforms respectively defined for home automation and healthcare assistance are presented in sections VI-A and VI-B. 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\(^8\).

A. 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 for controlling each appliance. 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 product/appartment lifecycle.

Particular InfoItems on each appliance can be accessed and controlled using QLM. Table I 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 remotely modify the fridge *Thermostat* as well as the ventilation *Program* (see rows 1 and 2 in Table I). 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 technique is selected by designating the controllers (fridge and ventilator) as $P_c$ (see Table I). 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 I). 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

\(^8\)http://dialog.hut.fi
Table I
LIST OF DATA SYNCHRONIZATIONS SET UP BETWEEN THE HOUSE DEVICES

<table>
<thead>
<tr>
<th>No.</th>
<th>InfoItem (appliance)</th>
<th>Functionality and permissible values</th>
<th>Man’s mobile</th>
<th>Woman’s mobile</th>
<th>Manufacturer DB</th>
<th>Fridge controller</th>
<th>Ventilator controller</th>
<th>Data synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Thermostat (Fridge)</td>
<td>Enable to adjust the indoor temperature of the fridge. Six thermostat values {1, 2, ..., 6} can be selected</td>
<td>✓ ✓ ✓</td>
<td>✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>A-P&lt;sub&gt;c&lt;/sub&gt; ⇔ P&lt;sub&gt;c&lt;/sub&gt;=fridge</td>
</tr>
<tr>
<td>2</td>
<td>Program (Ventilation)</td>
<td>Enable to select a ventilation program (13 in total)</td>
<td>✓ ✓ ✓</td>
<td>✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>A-P&lt;sub&gt;c&lt;/sub&gt; ⇔ P&lt;sub&gt;c&lt;/sub&gt;=ventilation</td>
</tr>
<tr>
<td>3</td>
<td>eSmart modes (Fridge+Ventilation)</td>
<td>Enable users to indicate whether they are at {Home, Safe at home, Away, LongAway}</td>
<td>✓ ✓ ✓</td>
<td>✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>A-U&lt;sub&gt;e&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

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<sub>e</sub> technique is selected (see Table I).

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 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 I and illustrated in Figure 10 through communications denoted by “3”.

The DS<sub>agent</sub> developed in this paper is used in addition to QLM in order to realize the three expected data synchronizations (cf. figurename 10). Section VI-A1 details the steps resulting from such an implementation considering synchronizations 1 and 2 (of type A-P<sub>c</sub>), while section VI-A2 details these stages considering synchronization 3 (of type A-U<sub>e</sub>).

1) Data synchronization 1 and 2 (A-P<sub>c</sub>): 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<sub>c</sub> is selected (IP address of the fridge is ticked as P<sub>c</sub>). 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<sub>agent</sub>, 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 for the fridge (i.e., 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”). 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...
is involved in an active synchronization of type A-Pc and the woman’s phone has not been designated as the P node. Accordingly, the request is automatically forwarded to the fridge, as depicted by arrow ⃝ 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.

2) Data synchronization 3 (A-Uc): 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 V-B, 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 subscription 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 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 (i.e., A-P and A-U) using the standardized QLM interfaces. The next section provides a similar case study but for health assistance purposes.

B. 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 [61], [62]). Only the cardiologist is allowed to retrieve information from the jogger database and to communicate it 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

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

Network protocol analyzer capturing the set of QLM messages received and sent by the fridge controller.
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 [63], [64] (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).

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

Figure 14. Healthcare environment and data synchronization between particular healthcare providers and companies.
health service providers, a synchronization of type A-U_e may be more appropriate. These different (or potential) diagnosis synchronizations are 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 VI-C focuses on data synchronization 2, i.e., the synchronization of the diagnosis/InfoItem named Abnormal_HRV that can take two values: YES or NO.

C. 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 [65]. In our case, both the cardiologist and neurologist can diagnose an abnormal HRV 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 section VI-A 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”). A pre-defined service in the neurologist system has been configured to notify the neurologist when the diagnosis list is updated[10]. This corresponds to the frame named “Alarm(0)” 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 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 “5” and “6” 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.

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

[10] A QLM write in our study is used to send the notification to a mobile phone.
The diagnosis list is then updated in the neurologist system from NO to YES, as highlighted in Figure 17. However, in our platform, 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.

D. Synthesis

The case study carried out on the 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.). In both scenarios, and especially in healthcare environments, we deliberately neglected issues of user’s privacy and security policies because the goal was to validate the feasibility of our proposals. The support of these traditional data synchronization mechanisms using QLM now provides a solid basis to propose more advanced services to users and professionals, as has been shown through a few examples (alarm triggers, data analysis, discovery of
information and services, etc.).

In this regard, new strategies for context-aware data distribution (including data synchronization) could further be developed to automatically set up the appropriate data synchronization between relevant actors and systems. Indeed, currently, the parameters specified in the \texttt{initiate()} or \texttt{remove()} function are entered by the user/engineer. Context-aware algorithms could envision to automatically set the appropriate parameter values according to the product context and environment, the user needs, and possibly other relevant information (for making the configuration setting transparent to the user/engineer). A recent and relevant survey on context aware data distribution in the IoT field made by Bellavista et al. \cite{66} provides convincing evidence on the necessity of such a new generation of data distribution. The authors state that “context data distribution strategies able to self-adapt autonomously depending on current management conditions is still an unexplored research field”.

VII. CONCLUSION AND DISCUSSION

Networks, systems, and products of the 21\textsuperscript{st} 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 or within their immediate surroundings, as well as interactions with people and mobile devices. This concept 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” (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 are 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.

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 lifecycle 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 product-related information among various and distinct lifecycle actors (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 $DS_{\text{agent}}$, 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: \begin{enumerate*}[i)]  \item support for \texttt{read} and \texttt{write} operations,  \item support for an \texttt{event-based} subscription mechanism with \texttt{callback address}, and  \item support for \texttt{asynchronous communications} (e.g., using a TTL parameter). \end{enumerate*} Therefore, if QLM fails in its attempt to become the \textit{de facto} standard for IoT data exchange, the conceptual models proposed in this paper could be easily integrated into other standards/protocols.

The $DS_{\text{agent}}$ 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 messaging. 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

\begin{itemize}
\end{itemize}


[61] J. Leber, Health insurer’s app helps users track themselves.


