QLM messaging standards: introduction & comparison with existing messaging protocols

Sylvain Kubler, Manik Madhikermi, Andrea Buda, Kary Främling

Computer Science building, Aalto University,
Department of Computer Science and Engineering,
P.O.Box 15400, FI-00076 Aalto, Finland
Corresponding author: sylvain.kubler@aalto.fi

Abstract. Recent advancement in web technology enabled the development of new Business-to-Business (B2B) infrastructures, e.g. based on the concept of Service Oriented Architecture (SOA). These infrastructures enable seamless information exchange among different stakeholders and complex business procedures. However, there is still a lack of sufficiently generic and standardized application-level interfaces for exchanging the kind of information required by such infrastructures. These interfaces must be as complete and flexible as possible to support changing organization needs and structures. Their development is an essential step to design future SOA services and to enhance product lifecycle management. The Quantum Lifecycle Management (QLM) messaging standards are proposed as a standard application-level interface that would fulfill such requirements. This standard is introduced in this paper and compared to existing ones. Several real-life implementations are presented to show why such messaging standards are needed and how flexible QLM messaging standards are.

Keywords: Service Oriented Architecture; Internet of things; Quantum Lifecycle Management; Messaging protocols; Intelligent product

1 Introduction

Today's enterprise architectures are based on several computing paradigms that have evolved over the years with the advent of the ubiquitous network era and the so-called Internet of Things (IoT) [2]. This gave way to complex heterogeneous enterprise system combinations to meet the enterprise expectations. However, as the system complexity increases, its infrastructure and maintenance costs also increase. Organizations are therefore looking for ways to minimize Information and Communications Technology (ICT) costs, while offering better products and services to users [16]. Finding the right balance (cost minimization *vs*. product/service enhancement) is a challenging task for organizations since their IT infrastructure has to continually provide the infrastructure and systems that support their business needs, even when they drastically change. This is particularly difficult because infrastructures and systems are often designed in isolated factions, thus limiting their openness and collaboration [26]. A first step has already been

taken to make systems more adaptable through the use of Electronic Data Interchange (EDI) technologies that allow B2B integration [15]. However, such systems are still limited in terms of *flexibility* and require higher maintenance costs [11]. Recent advancements in web technology (e.g. web 2.0) provided the opportunity to develop new B2B infrastructures that make it possible to perform both seamless information exchange among different stakeholders and complex business procedures by integrating different internal and external services [24]. In this regard, the concept of SOA is a good example since it enables to turn business applications into individual business functions and processes [21].

SOA is an architectural paradigm and discipline that may be used to build infrastructures enabling those with needs (consumers) and those with capabilities (providers) to interact via services across disparate domains of technology and ownership [23]. A SOA architecture is a form of distributed system architecture whose main properties are service abstraction, orientation and autonomously [8]. Although the potential of SOA has been widely recognized, there are still fundamental questions and issues that need to be addressed, e.g. no proper agreement on a common standard for data exchange between organizations, whether in terms of data structure or data communication, has yet been reached; new flexible business and communication interfaces for supporting the many changes of organization infrastructures and needs throughout their lifetime has yet to be designed [11]; developing new strategies for context-aware services, i.e. services able to self-adapt autonomously depending on current management conditions is becoming a glaring demand [26]. These challenges have to be addressed so as to increase the scope of SOA architectures, i.e. to provide architectures sufficiently flexible to be used in any domain of the Product Life Cycle (PLC), regardless the product context and environment. The notion of *flexibility* is the watchword to design future and dynamic SOA services and to enhance the product data management (i.e. product data interoperability, visibility, sustainability, manageability and security) [3].

This paper introduces a new messaging protocol named Quantum Lifecycle Management (QLM) messaging that aims to enable all kinds of intelligent entities[1] in and between organizations to exchange IoT information in ad hoc, loosely coupled ways. This proposal, currently under standardization, complements the existing portfolio of messaging protocols (e.g. JMS, oBIX…), which are all potential solutions for supporting SOA features. Each protocol has *pros* and *cons* that may limit their openness to new applications and new organization needs. In this regard, section 2 gives a more comprehensive view of the necessity to implement an appropriate messaging protocol to ensure data exchange interoperability, which is of the utmost importance to efficiently support SOA infrastructures. Section 2 also presents four messaging protocols (in addition to QLM) well recognized in the literature, whose primary goal is to meet the data exchange interoperability requirements, as QLM. Section 3 defines a framework used in our paper to compare QLM with these messaging protocols. Section 4 provides greater details on the two standards that compose QLM. Section 5 details two real-life industrial applications where product information are collected, exchanged and processed in and

[1] An "entity" might be a product, a device, a computer, a user or any other type of system that consumes or provides information.

between various organizations based on the QLM messaging standards. These applications help to show how important and flexible these standards are.

2 Data exchange interoperability

2.1 Two-level challenge

Data exchange interoperability can be achieved at the application level, which is a two-level challenge as illustrated in Fig. 1:

i) *Communication level integration:* necessity to provide infrastructures and mechanisms that support communications and transactions between distinct organizations, networks, applications and IT technologies,

ii) *Data level integration:* necessity to handle the changes in data media and formats that occur throughout the product and information life cycle,

Over the last decades, techniques, concepts and standards have emerged at both levels [4,20]. For instance, the ISO 16739 or UDEF are standards defined at the level of data integration. The focus of this paper is not given at this level but at the communication level.

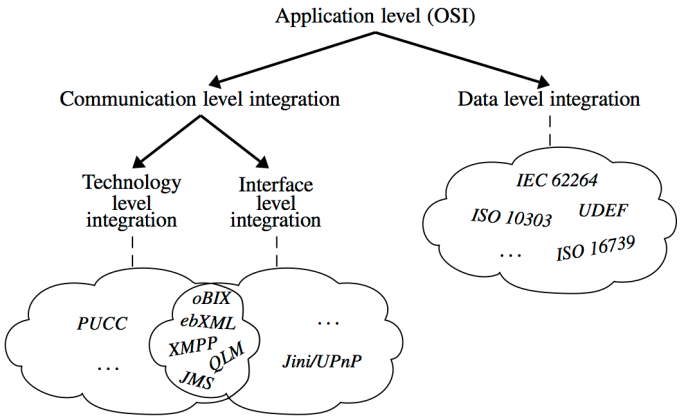


Fig.1. Data exchange interoperability at the "Application level"

The communication level integration is in turn a two-level challenge as depicted in Fig. 1:

i) *Technology level integration:* necessity to provide a framework for integrating heterogeneous hardware and software platforms and to support the wide range of IT technologies (proprietary solutions, extended networks, *etc*.),

ii) *Interface level integration:* necessity to support different types of transactions that must be based on shared business references [6]. To gain time and efficiency and to avoid re-defining co-operation rules and software support-

ing it each time, these references must be based on business standards or norms. The business standards must be independent and loosely coupled with the technology level integration so as to support openness.

It is not an easy task to develop solutions supporting both levels of integration. As a matter of example, the Peer-to-peer Universal Computing Consortium (PUCC) [18] mainly focuses on the technology integration aspect by designing a general peer-to-peer networking platform that enhances the communication capabilities of various devices via various networks. The solutions developed by this consortium do not integrate the set of interfaces directly at the communication level but requires the deployment of the PUCC middleware, thus limiting the integration with other middleware. Another example is the Jini/UPnP interoperability framework developed in [1] that allows clients to use pre-defined interfaces, but requires the use of Java technologies. This makes technology integration with non-Java applications quite daunting. A few solutions have nevertheless emerged for supporting both levels of integration such as oBIX (Open Building Information Exchange), ebXML (Electronic Business XML), XMPP (eXtensible Messaging and Presence Protocol), QLM (Quantum Lifecycle Messaging) or still JMS (Java Message Service). This paper only focuses on such protocols. The next section briefly introduces these five messaging protocols, which are then compared in section 3.

2.2 Candidate messaging protocols

JMS might be the most known messaging protocol among the five to be compared. The creation of JMS was joint effort of several corporations supported by SUN Microsystem [17]. JMS is an Application Programming Interface (API) for enterprise messaging that provides the set of interfaces enabling point-to-point communications in distributed enterprise applications. JMS defines a single, unified message API that enables loosely coupled, asynchronous and reliable communications, and support the development of heterogeneous applications that span operating systems, machine architectures, and computer languages. JMS relies on the publish/subscribe (Pub/Sub) model and supports the subscription mechanism. In this model, publishers and subscribers are generally anonymous and may dynamically publish or subscribe to specific data. A centralized system takes care of distributing subscribed data coming from the publishers to all subscribers.

oBIX is an industry-wide initiative in the building area that enables mechanical and electrical control systems to communicate with enterprise applications. oBIX defines XML and Web Services-based mechanisms and the information is represented as a concise object model, where objects with any structure and contents can be freely exchanged [22]. A client-server architecture is used for accessing oBIX information over the network. Software can act as a client, as a server or eventually as both simultaneously. Three types of requests/responses referred to as "oBIX verbs" are defined:

- *Read*: it returns the current state of an object,
- *Write*: it updates the state of an object,
- *Invoke*: it triggers an operation on an object.

oBIX also provides mechanisms called *watch objects* that enable clients to subscribe to object events. More concretely, *watch objects* are created based on client requests, then the server provides the client with a URI to enable it to poll and get the latest status updates. It makes it possible to append new history records, query historical data or do basic roll up of numeric data. Subscriptions are cancelled either when the client unregisters the *watch object* or when the predefined lease time for polling action is exceeded. The oBIX alarm feature is a cornerstone of this protocol, which enables to raise an alarm when an incident occurs and to notify users or related applications. Broadly, oBIX is a suitable communication interface for data acquisition and control of a wide range of devices.

XMPP is a communication protocol for message-oriented middleware based on XML [27]. XMPP was introduced by the Jabber open-source community in 1999 and has been improved by the XMPP Working Group to provide Internet Engineering Task Force (IETF) Instant Messaging and presence technology. The XMPP RFC defines this messaging protocol as a robust, secure, scalable, internationalization-friendly architecture for near real time messaging and structured data exchange. In XMPP, the communication of XML messages is based on the XML stream rather than the XML document between two network entities. The XMPP technology uses decentralized client-server architectures in which clients open a stream to the server, which in turn opens a stream back.

ebXML is an initiative of the OASIS group and the United Nations Centre for Trade Facilitation and Electronic Business, whose foundations rely on a former B2B solution called ooEDI (object oriented Electronic data interchange) [14]. This consortium promotes an open and XML-based infrastructure supporting the exchange of electronic business information in an interoperable, secure, and consistent manner. One objective for ebXML is to lower the barrier of entry to *e*-business, particularly in the context of small and medium enterprises. The ebXML specifications cover business processes and heterogeneous data shared between a wide range of actors and corporations.

QLM messaging standards emerged out of the PROMISE EU FP6 project[2] in which the real-life industrial applications required the collection and management of product instance-level information for many domains involving heavy and personal vehicles, household equipment, phone switches, *etc*. Information such as sensor readings, alarms, assembly, disassembly, shipping event, and other information related to the entire PLC needed to be exchanged between several organizations. The project consortium set out to find suitable standards for exchanging such information. PROMISE created two main specifications that fulfilled the necessary requirements: the PROMISE Messaging Interface (PMI) and the PROMISE System Object Model (SOM). At the end of the PROMISE project, the work on these standards proposals was moved to the QLM workgroup of The Open Group[3]. The QLM messaging protocol is the continuity of PMI and consists of two standards proposals [12], namely the QLM Messaging Interface (QLM-MI) that defines what kinds of interactions between entities are possible, and the QLM

[2] http://promise-innovation.com
[3] http://www.opengroup.org/qlm/

Messaging Format (QLM-MF) that defines the structure of the information included in a QLM message. Both standards are presented in section 4.

3 Messaging comparison framework

In our study, several key properties are considered to compare the five messaging protocols previously introduced. These properties mostly come from the framework defined in [28], which are divided into three main categories: *i)* message delivery model, *ii)* message processing model, *iii)* message failure model. The next three paragraphs respectively introduce the set of criteria defined for each of these categories.

3.1 Message delivery model

This category defines the properties related to the delivery of the message to the intended recipient. The majority of the criteria composing our framework comes from this category:

1. *Representation:* it indicates the message representation adopted by the messaging protocol. It can be represented either as a *i) Data element* - the message consists of a messaging header and body that respectively contains information about the interface and the data to be conveyed (e.g. an XML message); or as an *ii) Object* - the message is specified as an object, which is nothing more than a programming method where its arguments are the set of information that compose the message. The *Object* representation assumes that this object/method is held by both the sender and the recipient of the message, while the *Data element* representation does not,
2. *Messaging API:* the protocol might be application-specific or -independent,
3. *Initiation:* it defines how the message is transmitted between two nodes, which can be either server initiated (i.e. *push* mechanism) or client initiated (i.e. *pull* mechanism). In the *pull* mechanism, a client queries the server whenever it needs the data. The messaging protocol might support one or both mechanisms,
4. *Intermediation:* it specifies whether the messaging protocol offers the possibility to rely on an intermediate party to complete its transaction,
5. *Persistence:* it specifies whether the messaging protocol has a provision for data persistency. Two modes are defined: *i) Persistent* - the data is stored in the system until and unless it is retrieved, *2) Transient* - the data is stored in the system as long as the message is valid (i.e. until TTL expires),
6. *Subscription:* it specifies whether the messaging protocol proposes subscription mechanisms. Two mechanisms are considered: *i) interval-based* and *ii) event-based*. In some of these protocols, a callback address can be specified when creating the subscription, which is used as address pointer for sending new values of the subscribed data to the subscriber node,
7. *Self-Contained:* it specifies whether the message contains all the necessary information to enable the recipient to appropriately handle the message. In more

concrete terms, the message contains all the relevant information such as the actions to be performed (read, write, subscription…), the message validity period (TTL), the mode of communication (asynchronous or synchronous), the callback address, *etc*.,

8. *Protocol agnostic:* it specifies whether the messaging protocol supports multiple underlying protocols, making it possible to transport the message using most "lower-level" protocols such as HTTP, SOAP, SMTP, FTP or similar protocols. It might also be possible to transport this message using files on USB sticks or other memory devices.

9. S*ynchronicity:* it defines whether the messaging protocol supports s*ynchronous* and *asynchronous* communications,

10. *Delivery-Guarantee:* it defines whether the messaging protocol has provision for the delivery guarantee (e.g. by returning responses).

11. *Piggy backing:* it defines whether the messaging protocol allows piggy backing a new request with a response (i.e. without dissociating the new request of the response). This is a crucial property both for real-time communications and to enable communications with nodes located behind a firewall,

12. *Multiple payloads:* it states whether the messaging protocol supports multiple payload formats.

3.2 Message processing model

This category deals with the message reception, i.e. how messages are processed by the recipient node and how responses are returned to the requester. Two criteria are defined:

13. *Processing result:* it defines how the requested information is returned to the requestor node. Three modes of response are defined: *i) single return value* - for every request, a single response is generated, *ii) single integrated return value* - the response contains the integrated value for the requested data (mode required when supporting subscriptions without callback), *iii) set of individual return value* - multiple response at different intervals are generated (mode required when supporting subscriptions with callback),

14. *Communication:* it defines how two nodes communicate once the request message is received. Two levels of communication are defined: *i)* Separate message, *ii)* callback address.

3.3 Message failure model

The third category describes what provisions the messaging protocol provides in case of failures and disruptions. Only one criterion is defined:

15. *Failure notification:* it defines whether the messaging protocol has a provision for failure notifications. Three main approaches are usually implemented: *Timeout of Acknowledgement*, *Reply with error message* or *Exception*,

3.4 Comparison study

The five protocols previously introduced are compared in Table 1 based on the comparison framework. Results highlight that the QLM messaging protocol covers the vast majority of the properties and sub-properties (i.e. it often offers the possibility to choose between different options where possible). It can be observed that JMS also covers many of these properties, but fundamental ones like the *piggy backing* (property 11) or still the *Callback* functionality (property 14) are missing. However, such functionalities are of great importance to increase the scope of SOA applications.

The *piggy backing* functionality, which is only covered by QLM, is an important aspect to be considered in industrial applications. Indeed, organizations today take proactive measures to protect the security, confidentiality, and integrity of their data [5]. This inevitably leads to a challenging conflict between data security and usability; security making operations harder, when usability makes them easier. It is therefore necessary, in certain situations, to enable two-way communications through firewalls (e.g. for real-time control/maintenance) with the presence of conditions. The communication model consists to piggy back new QLM requests with the generated responses[4].

An aspect of prime importance considering complex product lifecycles is the support of *Multiple payloads* for being embedded in a message. The non-support of this functionality is particularly critical when many actors, corporations and systems are involved in the PLC. Indeed, the higher the product complexity, the higher the number of formats used to store information. In this regard, all protocols support this functionality.

Another important aspect is related to how protocol returns the *processed result*. Table 1 clearly states that QLM supports all possibilities (see property 13), while others do not. Once more, the faculty of QLM to propose a panel of possibilities proves its flexibility to face SOA environments and applications with diversified characteristics.

As emphasized in property 4 of Table 1, the *intermediation* layer is mandatory in the four protocols, while QLM provides the possibility to choose between using intermediation or not. This solves the issue of intermediation dependency and reinforces the claim that QLM messaging is highly appropriate for SOA implementations.

To conclude, let us add that the conceptual framework used in the QLM messaging protocol to subscribe a data (*cf.* property 6) is the Observer Design Pattern presented by [13] and applied according to [9] which signifies that a node can add itself as an observer of events that occur at another node. In this sense, QLM differs from e.g. JMS, which is based on the Publish/Subscribe (Pub/Sub) model. For many applications, the Observer and the Pub/Sub models can be used in quite similar ways. However, the Pub/Sub model usually assumes the use of a "high-availability server", which the Observer pattern does not [7]. This is why the Ob-

[4] The condition, in this case, is that the node that performs the piggy backing must first be contacted by the node located behind the firewall.

server model is more suitable for IoT applications where products might communicate with each other directly.

This comparison shows that the QLM messaging protocol covers a wide range of interfaces, which inevitably plays a leading role in making SOA techniques/algorithms sufficiently generic to be instantiated in new applications and systems, throughout the organization and product lifetimes [6]. Section 4 describes in greater details the QLM messaging protocol.

Table 1. Messaging protocol comparison based on 15 criteria

n°	Category	Property	Sub-property	oBIX	ebXML	XMPP	JMS	QLM
1	Message delivery model	Representation	Object				✓	
			Data Element	✓	✓	✓		✓
2		Messaging API	Application-specific	✓		✓		
			Application-indep.		✓		✓	✓
3		Initiation	Push		✓	✓	✓	✓
			Pull	✓	✓		✓	✓
4		Intermediation		✓	✓	✓	✓	✓[4]
5		Persistence	Transient				✓	
			Persistent	✓			✓	✓
6		Subscription	Interval-based	✓			✓	✓
			Event-based	✓	✓	✓	✓	✓
7		Self-contained		✓	✓		✓	✓
8		Protocol agnostic			✓		✓	✓
9		Synchronicity	Synchronous		✓	✓		✓
			Asynchronous	✓	✓	✓	✓	✓
10		Delivery-guarantee		✓			✓	✓
11		Piggy backing						✓
12		Multiple payloads		✓	✓	✓	✓	✓
13	Message processing model	Processing Result	Single return value		✓	✓	✓	✓
			Single integrated...	✓				✓
			Set of individual...	✓				✓
14		Communication	Message ID	✓	✓	✓	✓	✓
			Callback					✓
15	Message delivery model	Failure Notification	Timeout of Ack.				✓	✓
			Error message	✓	✓	✓	✓	✓
			Exception				✓	

[5] Unlike the four other messaging protocols, intermediation is not mandatory in QLM. However, QLM offers the possibility to use such functionality.

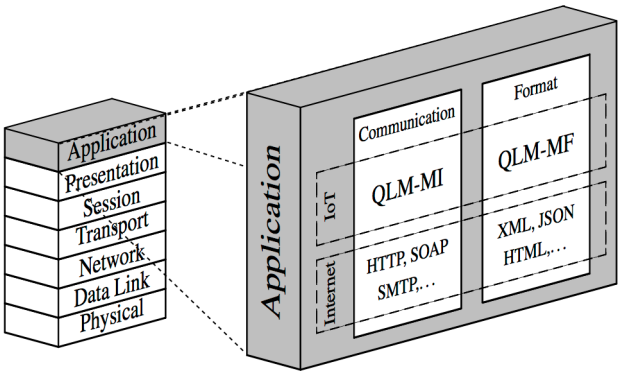


Fig.2. QLM-MI & QLM-MF in the OSI model

4 QLM messaging standards

The origins of QLM messaging standards come from the work on developing a universal information system architecture for the IoT based on an open-source middleware software called DIALOG and its underlying product agent [10] and intelligent product concepts [19]. However, during the PROMISE research project, it became clear that a purely open-source based approach was not sufficient in an environment with many organizations whose existing software needed to communicate PLC data between each other. This eventually led to the creation of the QLM standards initiative.

4.1 QLM Messaging Interface

In the QLM world, communication between QLM nodes is done by using interfaces defined in QLM-MI. QLM-MI and QLM-MF are independent components as illustrated in Fig. 2. QLM messaging standards reside in the Application layer of the OSI model, where QLM-MI is specified in as a *Communication level* and QLM-MF is specified in as a *Format level*. Where the Internet uses HTTP for transmitting HTML-coded information mainly intended for human users, QLM is used for conveying lifecycle-related information mainly intended for automated processing by information systems. In the same way as HTTP can be used to transport payloads also in other formats than HTML, QLM can be used for transporting payloads in nearly any format (*cf*. Fig. 2). XML might currently be the most common text-based payload format due to its flexibility and structuring that provide more opportunities for complex data structures [26]. However other payload formats such as JSON, CSV can be used. As studied in section 3, the QLM messaging protocol offers a wide range of properties, whether regarding QLM-MI or QLM-MF. A cornerstone property of QLM-MI is the concept of deferred re-

trieval of information, which corresponds to the subscription mechanisms. QLM-MI actually defines two variants of subscriptions[6]:

- *with callback address:* the data is sent using a QLM response at the requested interval. The interval "-1" means that it is event-based (i.e. the subscribed target has to push the value each time it publishes a new value),
- *without callback address*: the data can be retrieved (i.e. polled) by issuing a new QLM read query by specifying the subscription ID, which has been returned by the server when creating the subscription.

4.2 QLM Messaging Format

QLM-MF is defined as a simple ontology, specified using XML Schema, which is generic enough for representing "any" object and information that is needed for information exchange in the PLC, or in the IoT to be more generic. It is intentionally defined in a similar way as data structures in object-oriented programming as illustrated in Fig. 3. It is structured as a hierarchy with an "Objects" element as its top element (cf. row 6 of the XML message and the related-hierarchy in Fig. 3). The "Objects" element can contain any number of "Object" sub-elements (*cf.* row 7 and the hierarchy). The most important attribute of an "Object" is "type", which specifies what type of object is[7]. In this example, the messaging interface (i.e. QLM-MI) specifies that it is a write operation (see row 2), which concerns the object "Refrigerator" (see row 7) whose ID is given et row 8: *SmartFridge22334411*. In this example, only one object is contained by this message but, as mentioned previously, others might be included, such as the TV, the washing machine or any other object from the fridge's environment). "Object" elements can have any number of properties, referred to as InfoItem(s) (*cf.* row 9 and 12), which are in this example information about the fridge location and the thermostat value. An "Object" element can also have any number of sub-elements (e.g. a subpart of the fridge that should not be directly characterized by an "InfoItem" could, for instance, be characterized by an "Object").

In our example, the write request aims at changing the values related to the fridge location and its thermostat, but more complex scenarios and applications could get the most out of this Format proposal due to the high generality of the Objects tree. In the QLM specification, pre-defined specialized types of "Object" will be proposed as shell components, which will respect specific data structures or data models. For instance, the QLM Data Model that was initially conceived as the basis for the information model for the Product Data Knowledge Management/Decision Support System (PDKM/DSS) in PROMISE SOM, is now defined as a single object in the QLM specification, which can be used as an independent object in the hierarchy [25]. This makes the QLM messaging protocol truly independent of data models adopted by applications.

---

[6] A subscription request is a QLM read that includes, among others, an interval parameter in the message interface.

[7] An optional attribute called "udef" may be used for specifying the object class using the Universal Data Element Framework (UDEF; www.udef.com) taxonomy.

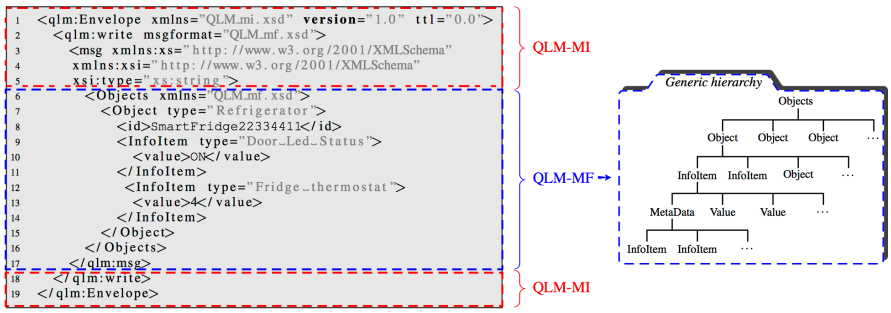


Fig.3. QLM message format

5 Application scope of the QLM messaging protocol

As stated previously, the QLM messaging protocol emerged out of real-life industrial applications where product information related to the entire PLC needed to be collected, exchanged and processed between several organizations. In this regard, QLM provides a wide range of interfaces and a flexible messaging format to let this happen. Fig. 4 illustrates how QLM enables to interconnect distinct organizations throughout the PLC, where various applications and actors take place. In this regard, two distinct scenarios are presented in sections 5.1 and 5.2. The first scenario is defined in BoL (Beginning of Life; *cf*. Fig. 4), while the second one takes place between MoL (Middle of Life) and BoL.

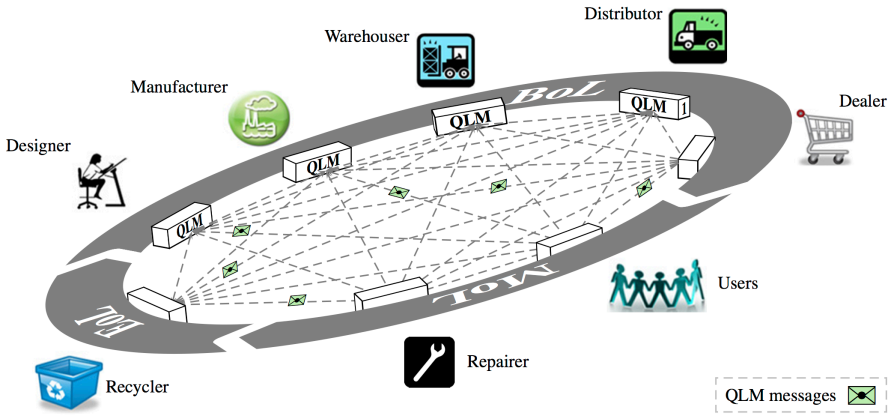


Fig.4. QLM architectures allowing data exchange between organizations

5.1 Production line of car chassis

This scenario is a real case study carried out in the framework of the LinkedDesign EU FP7 project[8]. In this scenario, different actors work on a production line of car chassis as illustrated in Fig. 5: chassis are first moved from the oven to a press machine, and then to other operations. This process segment involves two robots to transfer the chassis from machine to machine (see robots 1 and 2 in Fig. 5). The actors involved in the manufacturing plan expressed, on the one hand, the need to check each chassis throughout the hot stamping process and, on the other hand, the need to define communication strategies adapted to each actor. To comply with these two requirements, first, scanners are added between each operation for the verification procedure (see scanners 1 and 2), and then the QLM messaging protocol is adopted to provide the types of interfaces required by the different actors. In more concrete terms, physical nodes from the production line have been augmented with the QLM messaging capabilities as depicted in Fig. 5, namely:

- the two scanners,
- a server in charge of collecting all the events generated by the scanners,
- the PDA of the line maintainer,
- the computer of the quality manager.

The next two sections define the appropriate communication strategies regarding each actor, which correspond to the dashed lines in Fig. 5 annotated: ① – ④.

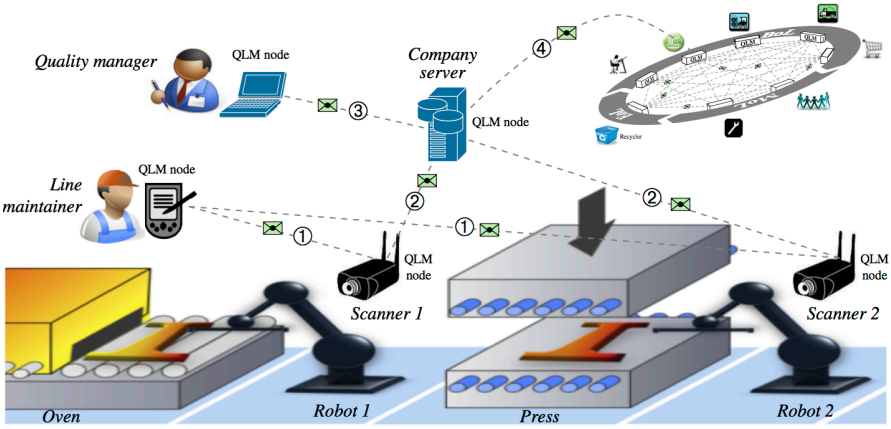


Fig.5. Hot stamping process monitoring and control (BoL)

Line maintainer The line maintainer expressed the need of receiving all the verification events generated by scanners 1 and 2 so as to identify in real-time whether a problem occurs on a chassis. Accordingly, the line maintainer directly subscribes

[8] http://www.linkeddesign.eu

(via his PDA) to the events generated by scanners 1 and 2 (*cf*. communications ① in Fig. 5). This is made possible by performing a QLM read query by *i)* setting the interval parameter to "-1" (this value means that the subscription is *event-based*[9]), *ii)* by including his own address as callback (i.e. the PDA's address) and iii) by setting the TTL parameter to "–1" (this value means that the validity period of the subscription is *forever*[10]). The parameters are defined in the message interface as shown in lines 1 and 2 of the subscription request provided in Fig. 6. Line 7 of this request indicates the name of the InfoItem that the manager subscribes, i.e. *StatusD*.

```
1  <qlmEnvelope xmlns="QLM_mi.xsd" version="1.0" ttl="-1">
2    <read msgformat="QLM_mf.xsd" interval="-1" callback="http:
         //207.46.130.1/ServletPDA">
3      <msg xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http:
         //www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">
4        <Objects xmlns="QLM_mf.xsd">
5          <Object type="Hot_stamping_machine">
6            <id>HotStamp1223</id>
7            <InfoItem class="StatusD"></InfoItem>
8          </Object>
9        </Objects>
10     </msg>
11   </read>
12 </qlmEnvelope>
```

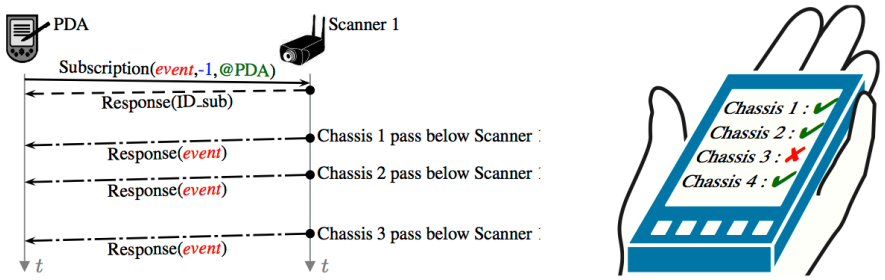
Fig.6. PDA subscribes (forever) to the InfoItem named *StatusD* on Scanner 1

The sequence diagram in Fig. 7(a) gives insight into the transactions resulting from this subscription. First, a response that contains the ID of the subscription is returned to the PDA. This ID is needed if the line maintainer wants to cancel this subscription. Then, each time a chassis passes under scanner 1, the subscribed InfoItem value (i.e. *StatusD*) is pushed to the PDA through a new QLM response. Based on these events, it is possible to develop scripts, for instance, to raise an alarm if a failure occurs on a chassis. Such an example is illustrated in Fig. 7(b) where chassis 3 has a default. However, the development of such scripts/tools is outside the scope of the QLM messaging protocol, but rely on the data obtained via that one. As the line maintainer subscribes to scanner events by specifying a callback address, the company server subscribes to these events in a similar manner (see communications ② in Fig. 5). Then, other people internal or external to the organization can access the subscribed data on the server.

Quality manager Unlike the line maintainer, the quality manager is not interested in receiving a continual flow of events from the scanners. His primary function is not to guarantee real-time control, but rather to deal with weekly or monthly evaluations (e.g. to estimate the failure rate over a period of time). Accordingly, the

[9] The node has to push the subscribed value every time this value is modified.
[10] The subscription is valid as long as the node initiator of the subscription does not cancel it.

second type of subscription supported by QLM is more appropriate in this situation, which consists in retrieving (i.e. polling) one or several historical values on the server (*cf*. communication ③ in Fig. 5) by issuing a new QLM read query containing the subscription ID. More concretely, the quality manager (via his computer) has to send a read query by setting the interval parameter to "-1" but, this time, without including a callback address, as illustrated with the argument "null" in Fig. 8 when the computer sends its subscription request to the server. Then, the targeted node (the server) is aware that the events generated by scanners 1[11] should be kept on the server as long as this subscription is valid. The quality manager can therefore issue a new read query by requesting one or several historical values, as depicted in Fig. 8.



(a) Sequence diagram related to the subscription (b) Triggering of the alarm

Fig.7. Subscription with callback address performed by the line maintainer

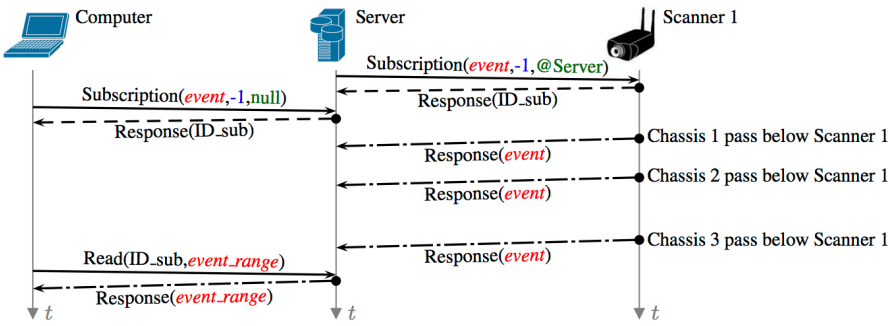


Fig.8. Subscription with callback address performed by the quality manager

External actors or organizations If other organizations in the PLC support the QLM messaging protocol and if the security rules allow them to access specific

[11] As mentioned previously, the server subscribed the InfoItem *StatusD* to scanner 1 and 2 beforehand, which is a subscription of type *based-event* as depicted in Fig. 8.

devices and information on the production line, they can further take advantage of the QLM interfaces to appropriately subscribe, modify or read specific device InfoItems. Such possibilities correspond to communication ④ in Fig. 5.

5.2 Smart house application

The scenario described in this section involves actors from two distinct PLC phases as depicted in Fig. 9, namely:

- in MoL: a user has equipped devices from his house (fridge, TV, *etc.*) with the QLM messaging protocol,
- in BoL: the fridge designer agreed with the user to collect specific fridge information over a certain period of the year (summer season),

The next two sections again describe the expectations of each actor and the appropriate QLM interfaces to set up.

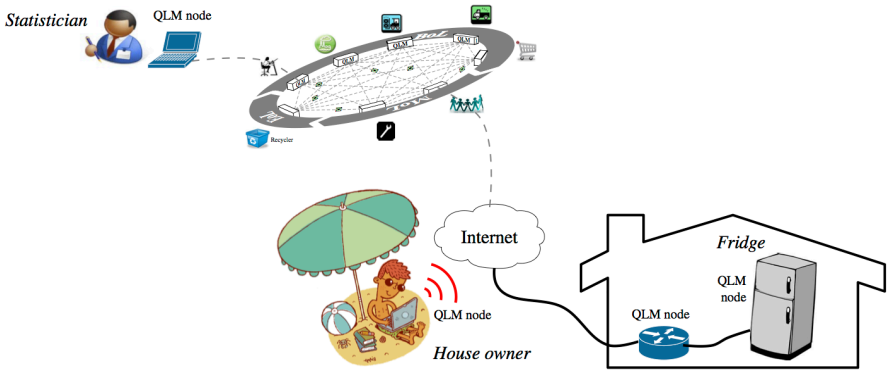


Fig.9. Smart house (MoL) and related service provider (BoL)

House owner The house owner goes on vacation for a period of two weeks and would like to continuously monitor the fridge temperature during this period. Since the owner is not aware of all the features of each device, the RESTful QLM "discovery" mechanism can be used to retrieve the exact InfoItem(s) to be subscribed to. An example of how this can be achieved by using the Unix *wget* utility is shown in Fig. 10 with *wget 1*, which returns the set of devices from the house that are reachable using QLM. Then, the user can refine his research by retrieving the set of InfoItems related to a specific device (e.g. regarding *SmartFridge22334412*). Such a refinement is performed via *wget 2* in Fig. 10, which returns the list of InfoItems that can be accessed (for read, write, subscription, *etc.*) on *SmartFridge22334412*. Then, the user is able to send a subscription request to this fridge by including:

- the InfoItem to subscribe: it is *RefrigeratorIndoorTemperature*,
- the callback address: required by the fridge to send the subscribed value,

- the interval parameter: the user does not want to perform a subscription based-event as done in the previous scenario, but would like to get the indoor temperature every hour. Accordingly, the interval parameter is set to 3600 (expressed in seconds) in the message interface,
- the period of the subscription validity: this corresponds in our scenario to two weeks. Such a period is specified through the TTL parameter contained in the message interface.

Fig. 11 shows the evolution of the fridge temperature over the two weeks after subscribing it.

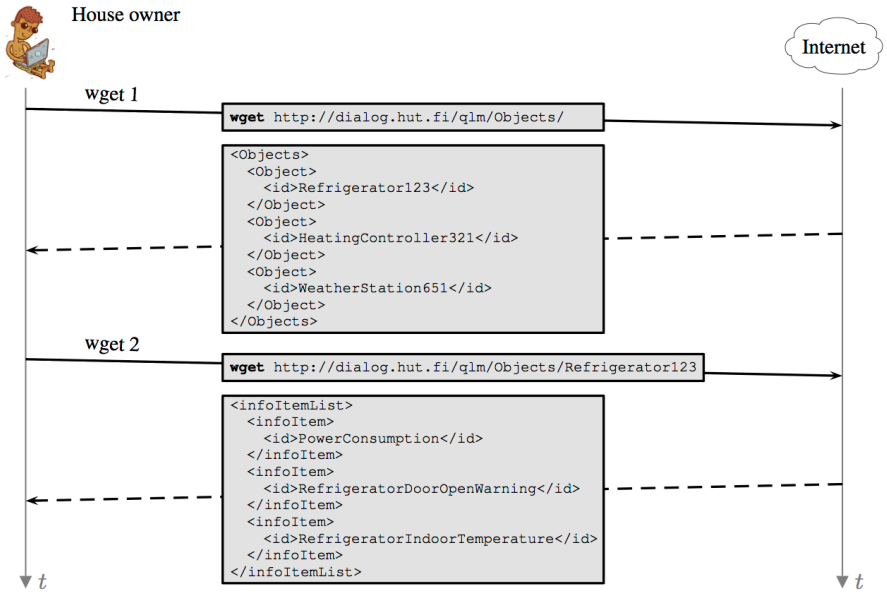


Fig.10. RESTful mechanism used by the owner to retrieve device information

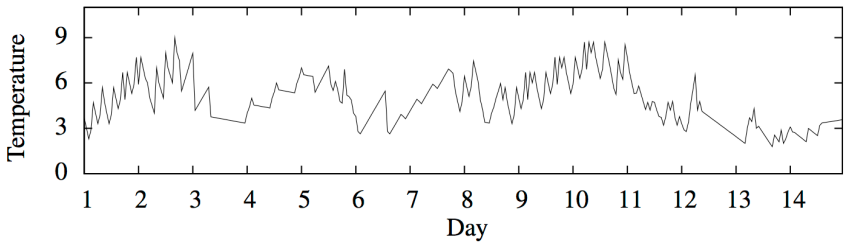


Fig.11. Evolution of the fridge temperature resulting from the subscription

Fridge designer As with the quality manager in the previous scenario, the statistician is not interested in receiving a continual flow of events from the smart fridge. Rather, the statistician is interested in retrieving historical values over a period of time considering a panel of users, which will help to develop learning models and algorithms capable of representing the fridge behavior into diversified environments and, accordingly, to enhance the design of the future generations of fridges. The statistician thus performs a subscription with a specific interval, without call back address and with TTL equal to 8035200 seconds. Indeed, as explained previously, this value corresponds to the validity period of the subscription which, in our case, corresponds to 3 months (June, July, August): $\approx 3 \times 31 \times 24 \times 3600 = 8035200$ seconds. Once the subscription is created, the statistician is able to retrieve a range of values over this period by sending a classical read query that includes the subscription ID (similar to that in Fig. 8).

6 Conclusion

New challenges and opportunities arise with concepts such as Internet of Things (IoT), where objects of the real world are linked with the virtual world, thus enabling connectivity anywhere, anytime and for anything. The rapid expansion of the IoT and the web technology enabled the development of new Business-to-Business (B2B) infrastructures based on the concept of Service Oriented Architecture (SOA). The technical provision of services is not a uniform activity but is skewed by prevalent technological solutions, which have often a lack of interfaces required by users or are designed in an isolated faction that limit their openness. Such a limitation has a direct impact on organizational infrastructures that, in today's world, need more than ever to exchange product information in an appropriate manner (e.g. to provide the right service or information, whenever it is needed, wherever it is needed, by whoever needs it).

The Quantum Lifecycle Management (QLM) messaging standards are proposed as a standard application-level interface that would provide flexible and a wide range of properties/interfaces, which aim to increase the SOA scope as well as the data exchange interoperability in the IoT. The QLM messaging properties are introduced in this paper and compared to properties of existing messaging protocols (e.g. JMS, oBIX, ebXML…). This comparison study shows that QLM covers more properties than those solutions, including fundamental ones like piggy backing, callback functionality or still the support of multiple payload formats. Several real-life implementations are also presented in this paper, which show how flexible the QLM messaging protocol is in the framework of Product Lifecycle Management.

Too often, SOA infrastructures give little attention to the hardware architecture by using "closed" technologies and applications (e.g. using private technologies or adding new software codes). Such designs both limit their interoperation with other applications and hinder their transposition to other domains/sectors. Messaging protocols such as QLM or similar ones (JMS, oBIX) are an essential step to design future SOA services and to enhance product lifecycle management.

For instance, further research could be carried out to provide context-aware SOA, which will help to develop ideas for new environment-friendly products, or still to provide customized and advanced products and improve the customer experience.

References

1. Allard, J., Chinta, V., Gundala, S., Richard III, G.G.: Jini meets UPnP: an architecture for Jini/UPnP interoperability. In: Symposium on Applications and the Internet. pp. 268-275 (2003)
2. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A survey. Computer Networks 54(15), 2787-2805 (2010)
3. Bazjanac, V.: Building energy performance simulation as part of interoperable software environments. Building and Environment 39(8), 879-883 (2004)
4. Berre, A.J., Hahn, A., Akehurst, D., Bezivin, J., Tsalgatidou, A., Vermaut, F., Kutvonen, L., Linington, P.F.: State-of-the art for interoperability architecture approaches. InterOP Network of Excellence-Contract no.: IST-508 11 (2004)
5. Bishop, M.: What is computer security? IEEE Security & Privacy 1(1), 67-69 (2003)
6. Chen, D., Doumeingts, G., Vernadat, F.: Architectures for enterprise integration and interoperability: Past, present and future. Computers in industry 59(7), 647-659 (2008)
7. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys 35(2), 114-131 (2003)
8. Feuerlicht, G., Govardhan, S.: Soa: Trends and directions. In: Proceedings of the 17th International Conference on Systems Integration. pp. 149-154 (2009)
9. Främling, K, Ala-Risku, T., Kärkkäinen, M., Holmström, J.: Design Patterns for Managing Product Life Cycle Information. Communications of the ACM, 50(6), 75-79 (2007)
10. Främling, K, Holmström, J., Ala-Risku, T., Kärkkäinen, M.: Product agents for handling information about physical objects. Report of Laboratory of information processing science series B, TKO-B, 153(3) (2003)
11. Främling, K., Holmström, J., Loukkola, J., Nyman, J., Kaustell, A.: Sustainable PLM through intelligent products. Engineering Applications of Artificial Intelligence 26(2), 789-799 (2013)
12. Främling, K., Maharjan, M.: Standardized communication between intelligent products for the IoT. In: 11th IFAC Workshop on Intelligent Manufacturing Systems, São Paulo (Brazil). vol. 11, pp. 157-162 (2013)
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Reading: Addison Wesley Publishing Company (1995)
14. Gibb, B.K., Damodaran, S.: ebXML: Concepts and application. John Wiley & Sons, Inc. (2002)
15. Gunasekaran, A.: Agile manufacturing: enablers and an implementation framework. International Journal of Production Research 36(5), 1223-1247
16. Haller, S., Karnouskos, S., Schroth, C.: The Internet of Things in an enterprise context. In: Future Internet, pp. 14-28. Springer (2008)
17. Hapner, M., Burridge, R., Sharma, R., Fialli, J., Stout, K.: Java message service. Sun Microsystems Inc., Santa Clara, CA (2002)
18. Ishikawa, N., Kato, T., Sumino, H., Murakami, S., Hjelm, J.: Pucc architecture, protocols and applications. In: 4th IEEE Consumer Communications and Networking Conference. pp. 788-792 (2007)
19. Kärkkäinen, M., Holmström, J., Främling, K, Artto, K.: Intelligent products--a step towards a more effective project delivery chain. Computers in Industry, 50(2) (2003)

20. Koronios, A., Nastasie, D., Chanana, V., Haider, A.: Integration Through Standards - An Overview Of International Standards For Engineering Asset Management. In: 4th International Conference on Condition Monitoring. pp. 11-14 (2007)
21. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R., Hamilton, B.A.: Reference model for service oriented architecture 1.0. In: OASIS Standard (2006)
22. Neugschwandtner, M., Neugschwandtner, G., Kastner, W.: Web services in building automation: Mapping KNX to oBIX. In: 5th IEEE International Conference on Industrial Informatics. vol. 1, pp. 87-92 (2007)
23. Nickul, D., Reitman, L., Ward, J., Wilber, J.: Service oriented architecture (SOA) and specialized messaging patterns. In: Adobe Systems Incorporated White Paper (2007)
24. O'reilly, T.: What is web 2.0: Design patterns and business models for the next generation of software. Communications & strategies 65(1), 17-37 (2007)
25 Parrotta, S., Cassina, J., Terzi, S., Taisch, M., Potter, D., Främling, K.: Proposal of an interoperability standard supporting PLM and knowledge sharing. In: Advances in Production Management Systems. Sustainable Production and Service Supply Chains, pp. 286-293. Springer (2013)
26. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Context aware computing for the Internet of Things: A survey. IEEE Communications surveys & Tutorials (99), 1-41 (2013)
27. Saint-Andre, P.: Extensible messaging and presence protocol (XMPP): Core. Tech. rep., Core; IETF: Fremont, CA, USA (2004)
28. Tai, S., Rouvellou, I.: Strategies for integrating messaging and distributed object transactions. In: IFIP/ACM International Conference on Distributed systems plat- forms. pp. 308–330 (2000)