Data interoperability assessment, case of messaging-based data exchanges

Jannik Laval, Nawel Amokrane, Boubou Thiam Niang, Mustapha Derras, Néjib Moalla

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE - TECHNOLOGY

Data Interoperability Assessment, Case of Messaging Based Data Exchanges

Jannik Laval*[1] | Nawel Amokrane[2] | Boubou Thiam Niang[1,2] | Mustapha Derras[2] | Néjib Moalla[1]

[1]DISP-UR4570, Univ Lyon, Univ Lyon 2, INSA Lyon, Université Claude Bernard Lyon 1, 69676 Bron, France
[2]DRIT, Berger-Levrault, Lyon, France

Correspondence
*Jannik Laval Email: jannik.laval@univ-lyon2.fr

Present Address
University Lumière Lyon 2, 160 Boulevard de l'Université, 69500 Bron

Summary

Data interoperability implies data exchanges among intra and inter enterprises collaborating with Information Systems (IS). The multiplicity of these exchanges and the increasing number of data exchanged generates complexity and brings out the needs for control to avoid dysfunctions with a negative impact on the overall performance of the systems. Indeed, actually, interoperability has become a necessary performance lever that thus requires particular attention. Being at a low level in the enterprise interoperability concerns, data interoperability is mainly automated, which leads us to question: is it possible to evaluate data interoperability performance and security based on inspection and analysis of ongoing data exchanges? We therefore endeavored to answer this question by establishing monitoring and analysis systems. In this paper, we present a research work which addresses services provided by a messaging based communication system. In order to collect information on Information System interactions allowing one to assess their level of data interoperability, we propose a Messaging metamodel that aggregates the collected information. It provides a single point of control and enables one to determine indicators of potential interoperability problems. The approach is validated on two case studies. An industrial case study of interactions among existing systems is presented to showcase the feasibility and interest of the approach. It is proposed on top of RabbitMQ and allows our partner to identify some issues in the studied information system. The second case study shows that the approach can integrate other protocols, by reading MQTT messages. The approach is implemented using Moose, a software analysis platform.

KEYWORDS:
Event-driven Architecture, Message Broker, Data Monitoring, Interoperability Assessment

1 | INTRODUCTION

Information systems are revolutionizing and transforming the activities of organizations. They are essential to any company in the process of implementing a strategy to achieve its objectives. O'Brien and Marakas [1] define an information system as an organized set of resources of various types. It is a combination of people, software, communication networks, hardware, data, policies and procedures that store, retrieve, transform and disseminate a variety of information within an organization [2].

In today's world, a company needs an information system to track all its activities. These activities, represented by business processes, can be supported by software tools and enterprise applications that aim to automate exchanges. It can be a source of competitive strength if it allows

the company to innovate or perform tasks faster than competing companies. Shackelford et al.[3] state that information systems specialists focus on integrating IT solutions and business processes to meet the information needs of companies, thus enabling them to achieve their objectives effectively and efficiently. However, the information system is often not perfectly aligned with the company's processes. On the one hand, the company's needs are constantly evolving and the information system must adapt to its requirements. On the other hand, the gap between specifications and implementation is one of the main causes of IT project failures[4]. In order to address these gaps, agility is seen an essential feature of information systems[5,6].

At the same time, the information system depends on more and is more numerous and interconnected software applications, and therefore more complex to transform. The information system is supported by interconnected computer applications. Its components are able to share and exchange information without depending on a particular actor and function independently of each other. We call them interoperable systems[7]. Interoperability is a property that defines the ability of two or more systems or components to exchange information and use the information exchanged [8]. Interoperability is also characterized as the relationship of exchange and cooperative use of information. In practice, the success of the cooperation depends heavily on the effectiveness of the interoperability between the participating systems. According to [9], data interoperability is "the ability of data (including documents, multimedia content, and digital resources) to be universally accessible, reusable, and comprehensible by all transaction parties (in a human-to-machine and machine-to-machine basis) by addressing the lack of common understanding caused by the use of different representations, different purposes, different contexts, and different syntax-dependent approaches". When established between communicating information systems, it ensures increasing productivity and efficiency of inter- and intra-company processes. This is technically possible through the automation of information exchanges based on the use of shared exchange formats and appropriate communication protocols. To achieve data interoperability[10], companies implement flexible, scalable and loosely coupled architectures, such as service-oriented and event-driven architectures (SOA, EDA).

A software application is defined as a living system[11]. Companies must take into account the new technological developments that regularly occur to meet the various business, technical (more reliable architecture, cloud, CPS, etc.) or qualitative (semantic integration, security, etc.) needs[12,13]. Beyond a "simple" software evolution, this transformation involves architectural, material and strategic changes, as well as processes and security modifications[14]. Despite these developments in software applications, the complexity continues to increase, particulary by shifting the focus from architecture to deployment issues[15]. Among the structure, interactions and data exchanges are concerned. Interoperability assessment methods take into account barriers that hinder enterprises and systems to interoperate. Even if some or all barriers are overcome, technical problems may arise due to the inherent nature of information systems and the constant evolution of systems that may alter the properties that have been set up to perform interoperability. However, even if interoperability is guaranteed by design, we noted the problems of stability and reliability of behavior when the constituents evolve: each constituent has organizational independence and can be added or removed from the system. Incidents may appear on the whole system, or emerging behaviors may arise. This dynamicity does not allow the overall behavior of the system to be anticipated at the time of its design[16]. Processes, interfaces may change, and the way interoperability is conducted may be affected. This is why it is important to monitor data exchanges and provide support to assess their performance. In this context, assessing the consequences of the lack of interoperability between systems is a critical issue and is considered as one of the key challenges for enterprise information systems interoperability research today.

In this article, we consider the dynamics of a distributed information system. The information system can become unstable or fragile, and lose its reliability. Our work here deals with the monitoring of existing data exchanges among distributed information systems. The proposed framework collects system data, and the latter is structured through Model Driven Engineering. This makes it possible to analyse and to propose elements of remediation. Our contribution is the result of a collaboration with Berger-Levrault. This article is an extension of previous articles[17,18] with a detailed explanation of the process, of the metamodel and more examples in the case study section.

2 | RELATED WORK

The system operation phase lifts many scientific barriers. Maturity models[19,20,21,22,23,24] were first used as approaches to assessing interoperability. They allow interoperability to be assessed against a set of predefined levels and provide recommendations for moving from one level to the next one in order to achieve the required level or reach full maturity. However, maturity models do not provide a precise indication of the causes of non-interoperability and mainly focus on general qualitative notions[25].

Once the distributed information system is put into production, its components are able to share and exchange information without depending on a particular actor and operate independently of each other, we speak of interoperable systems[7]. Interoperability has become an important business asset and is now proposed as a key performance indicator for business process performance management systems[26].

The problem of interoperability between different heterogeneous systems already exists and is amplified by the strong deployment of distributed applications and the Internet of Things (IoT). To meet this challenge, companies are emphasizing the use of open standards for data

format and communication protocols. Also, they can approach the problem using agile software development process such as Continuous Software Engineering [27,28] to increase the deployment frequency and the release delivery.

Some data interoperability issues are load-related. Interoperable systems can exchange large amounts of data while requiring low latency. This is the case when interacting with IoT systems [29]. The increase in the volumes of data exchanged implies the implementation of exchange architectures capable of supporting not only the load, but also the variability in the frequency of data production. This requires distributed architectures (both in terms of infrastructure and flows) that can be adaptable or even self-adapting [30], in order to strengthen the system's resistance to malfunctions while avoiding potential congestion phenomena and thus, guarantee reliable interactions in terms of interoperability.

Service Oriented Architecture (SOA) is one of the approaches used to integrate legacy platforms, protocols and systems [31]. It is characterized by simplicity, flexibility and adaptability. Researchers believe that SOA provides significant benefits to organizations, enabling them to dynamically build new applications and respond to changing business needs. Services represent the functionalities of the business [32]. As an architectural approach, it breaks down business applications into individual processes and services. They can be recomposed to create alternative applications. They can also be exposed to other systems allowing different applications to reuse common parts.

Event-driven architectures (EDAs) are architectures that implement services that react to external events. The use of events allows the architecture to operate asynchronously and with low coupling. An event-oriented architecture is based on a bus with subscription and publication features. Message brokers are typically used to decouple distinct stages of software architecture. They enable asynchronous communication, which promotes the decoupling of applications, using the subscription/publication paradigm. These message brokers also find new applications in the area of IoT devices and can also be used as a method to implement an event-driven processing architecture. The multiplicity of this type of data exchange generates complexity and control requirements.

Collecting event log data to understand or reengineer a complex process is known as Process Mining. This technique has been used for some years in software engineering [33,34,35]. It is applied to various software reengineering topics such as bug fixing [36]. Process mining is mainly used to reverse engineering of a work process. Given the dynamics and reliability analysis, this work is not suitable and needs to be coupled with a dynamic process analysis system and semantics to facilitate the analysis of interoperability problems.

In terms of use of event data, the field of process discovery is a closer match. From the event data, it provides abstractions to model the process as it works and links the logs to this abstraction [37]. These abstractions allow the process manager to optimize performance issues, using complex algorithms such as artificial intelligence [38]. The process discovery domain is suitable for identifying an existing process that is not documented or that must be understood. It does not consider the reliability of the system, which should be done by an expert when the discovery has yielded results. Again, the various works in this area need to be coupled with analysis systems to dynamically analyse the interoperability of a system.

In this article, we discover the architecture of a running system using the event log of the system itself. The closest existing approach we have found is the one of Gomez et al. [39]. It provides a metamodel for building an IoT system. The difference between this approach and the one presented in this article is that we evaluate running systems, while the objective of the other approach is to build a system. Similarly, for supervising an information system, we have developed an approach that allows us to identify architectural problems such as interoperability problems.

3 | OBJECTIVES

Event-driven architectures (EDAs) [40,41,42] allow loose coupling, but limit the ability to understand the overall behavior of the architecture. EDA is based on the principle that an event is sent by a publisher node to a broker on a topic without knowing which node will consume the message. A consumer subscribes to a topic and receive all messages published on this topic (Figure 1). The flexibility offered by this structure hide the path taken by the messages, and the understanding of the architecture. The multiplicity of data exchanges generates complexity and highlights supervision needs that can be met by setting up monitoring and analysis systems [43]. A monitoring system is defined as a distributed process or set of processes comprising the function and dynamic collection, interpretation and processing of information about a running application.

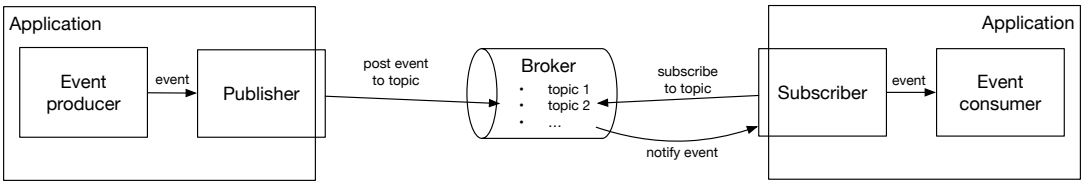

FIGURE 1 An EDA message flow

Existing structures (for example Apache Camel[1] [2], NServiceBus[3]) provide monitoring consoles. However, they mainly focus on low-level monitoring information such as message frequency, performance indicators or memory usage. RabbitMQ[4] provides a management console with information about the structure of the messaging system and the status of messages [44]. It presents lists of existing resources (channels, exchanges, queues ...), their contents, their characteristics and a set of statistics. It is possible, for example, to access queues and check pending messages. In our experience, the RabbitMQ console can be used for real-time monitoring and is suitable for specific requests where the maintenance manager knows which queues or exchanges to monitor. However, it does not allow advanced searches and filters on sources and messages in transit. Consumed messages are no longer visible in the management console. In addition, messaging channels such as exchanges, queues and their links are volatile and can be deleted when the consumer logs out.

The absence of these controls makes it difficult to analyse and diagnose dysfunctions, such as publisher and consumer inactivity, invalid exchange formats or unavailability of data. As a result, it is difficult to identify the context and origin of the problem, based solely on a management console. Our approach uses monitoring to perform an analysis capable of defining a kind of classification of the potential causes of problems occurring in trade, in order of importance and for a given problem. In the first study, we advocate that the data exchange behavior between communicating applications gives an indication of the level of interoperability of their data. A monitoring system should provide elements to maintain a good level of interoperability based on the interoperability requirements [25].

We therefore propose to combine the information provided to perform advanced monitoring and querying. This would provide indicators which determine maintenance actions. The main research problem addressed is to ensure the supervision of a distributed information system based on the metadata of the messages, transiting through the system and the metadata of the message broker. The research objectives we have addressed are the following:

1. Collect metadata of the messages transiting through the information system in order to analyse the architecture of the exchange system. For this purpose, we proposed the analysis framework called Pulse. It collects metadata in order to (i) track the messages exchanged, (ii) simplify the visualization of exchanges, (iii) improve maintainability by detecting exceptions (e.g. problems in the transfer of a message), and by specifying the context and the origin of the problem.

2. Organize the collected metadata so that they can be processed in a consistent way. To this end, we have proposed a generic and extensible metamodel adapted to the EDAs. It is capable of representing the AMQP and MQTT protocols (shown in the case studies) but also KAFKA and CoAP protocols based on a correlation explained in Table 2, and can be extended to other protocols. The messaging data model describes the messaging structure implemented via message queuing and the exchange system. It is used for collecting the metadata from the logging services offered by the exchange infrastructure and tracks the messages exchanged.

3. Identify interoperability issues and help solve them. To do this, we carried out a study of the causes and effects of the problems while identifying the indicators related to interoperability.

4. Take into account the lifecycle of the various components. To achieve this, the Pulse structure integrates dynamics modeling functionalities, where the life cycle of different components of the architecture is described including a history and the creation and deletion dates of the components.

4 | TERMS AND VOCABULARY

We address the supervision of distributed information systems based on four protocols:

- AMQP [45] : it's an asynchronous communication protocol through a mediator. It aims to create an open standard for transporting information between applications. A message is published in an exchange by a message producer, then routed by the message broker to one or more queues according to predefined routing keys. Consumers connect to the queue to retrieve the message.

- MQTT [46] : it's a communication protocol dedicated to the IoT. It was designed as an extremely lightweight asynchronous communication protocol. It can work in the same way as AMQP. The subscription / publication model allows dissociating the producer sending a message from the consumer receiving the messages. Compared to AMQP, MQTT includes a quality of service parameters. The level of service determines the type of guarantee that a message has to reach the desired recipient.

[1] http://sksamuel.github.io/camelwatch/
[2] http://rhq-project.github.io/rhq/
[3] https://particular.net/nservicebus
[4] https://www.rabbitmq.com/

- Kafka[47] : This protocol has been optimized to disseminate data between systems and applications as quickly as possible and in a scalable manner. It is also based on a subscription / publication communication model. The data are organized into subjects. Subjects are multi-subscribed, divided into partitions and each message broker can have one or more of these partitions.

- CoAP[48] : it is a specialized communication protocol for constrained devices, defined in RFC 7252. One of the main objectives of CoAP is to design a generic protocol for the needs of constrained environments, including energy, building automation, and machine-to-machine (M2M) applications. CoAP uses the UDP protocol.

In this article, we use some specific terms related to these protocols presented in Table 1.

TABLE 1 The Pulse terms associated with the protocols concepts

Pulse Terms	**definition**
Connection	A TCP network connection between an application and the message broker.
Channel	A communication flow between two constituents.
Message	A message consists of a header and a body. The header contains the properties of the message presented in a specific format type. The body or load is the application data also presented in a specific format type.
Exchange	A named entity that receives messages from producers and routes them to the queues.
Queue	An entity that contains the messages and delivers them to consumers.
Routing key	A virtual address that an exchange can use to route messages to specific queues.
Publisher	A client application that publishes messages for exchange.
Consumer	A client application that requests messages from the queues.

5 | THE PULSE FRAMEWORK

In order to collect, analyse and identify communication problems, we have defined a framework, illustrated in Figure 2. The monitoring framework is composed of four levels: (i) a data import level, (ii) a historical record management and model version management level, (iii) a persistence level, (iv) an analysis level.

The data import level, the persistence level and the analysis level are based on the Meta-Model presented in Section 6. They do not represent a strong problem. For importing data from different sources with different formats, we have defined dedicated importers that consume the messages or logs and instantiate the model with the collected information.

For the persistence, the historical record module keeps several versions of the model in the runtime environment. This is related to references between entities of different versions that are not copied from one version to another. When the persistence module is called, all versions of the Orion model are stored in a file. It can be extended to produce other types of structured data. Thanks to this feature, external system such as Grafana[5] can be used to calculate metrics and to display certain visualizations.

For the analysis level, the implementation of the historical record management allows one to both analyse and visualize the changes made to the monitored system in real time, as well as return to a previous state of the system to view and analyse the changes and their impact. These two features provide the detection of interoperability problems as they occur, and also allows one to analyse the potential source of a problem by returning to previous states.

[5]https://grafana.com/

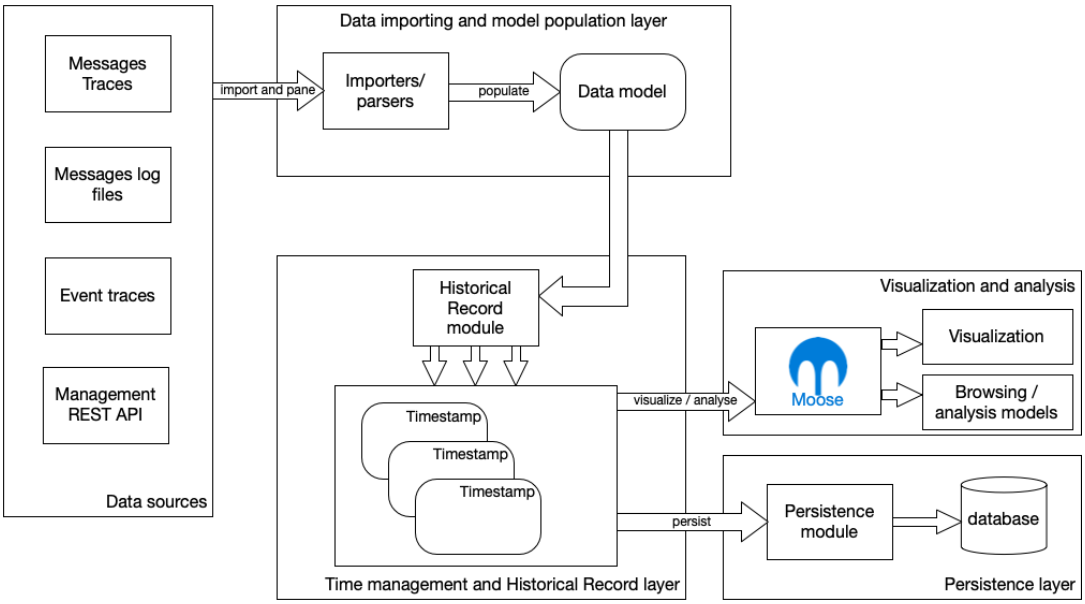

FIGURE 2 The Pulse Framework presented in a previous article [18]

The main feature is the Historical Record Management. When importing data from different sources, it is difficult to represent the dynamic aspects of the system. For example, the RabbitMQ architecture and its messages are volatile and therefore not visible at the management console level. We have therefore developed a kind of historization in order to understand previous events and facilitate a better analysis.

A trivial method could be to incorporate a timestamp for the creation, deletion and updating of each entity in the model. The problem with this approach is the strategy used for creating a specific status at a given time. Another method may be to create a model each time, which would be too space-consuming with each new piece of data coming from the analysers.

We've considered an alternative, based on Orion [49]. It is a model that allows one to create different versions of a data model, taking into account the tracking of changes made to that model. The basic principle of Orion is that each change triggers an Orion action to add it to the data model. Each change may result in a new version of the data model. Orion optimizes the persistence of different versions of the model, while managing deltas and pointers to previous versions. Orion copies only the entities affected by a change.

Figure 3 illustrates the version management strategy. A version of Orion includes new changes and information on the action taken to create that version. This version management of the data model allows one to track the evolution of the messaging architecture over time, where each version represents a snapshot of the architecture at a given point in time. For the user, each version represents a screenshot of the monitored system at a specific time. We have defined a strategy to create a version whenever necessary. In the case of the message exchange system, we have defined two types of events:

- A version of Orion is created when a change in the architecture or configurations/parameters of the supervised system occurs (creation of a queue, deletion of a queue, changed user rights, etc.). The status of the system before and after the change must be kept.
- A dedicated entity in the current version of the template is instantiated when a new trace (new message published, message received, new connections, etc.) appears.

6 | PULSE METAMODEL

The objective of the Pulse metamodel is to represent three aspects of the information system:

- A static representation: the architecture of the system implemented, representing the queues and channels.
- A dynamic representation, where messages are shown from the source component to the destination. It represents connections, messages, and any volatile entities.
- The lifecycle representation of the architecture, where components (e.g. queue, exchange...) are created, modified, deleted.

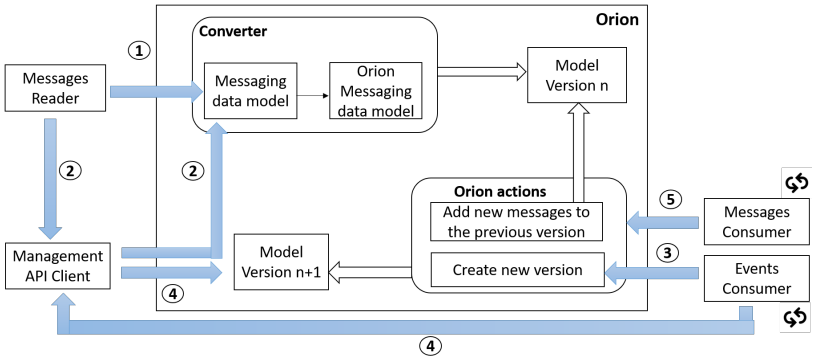

FIGURE 3 Historical Record management process with Pulse

Several protocols can be used in the same information system, one of the objectives of this metamodel is to be generic enough to take into account different protocols, and to be extensible. After presenting a first metamodel [17] based on AMQP, we have analysed the official documentation of MQTT, Kafka and CoAP protocols and we have compared the differences to create a metamodel allowing us to integrate them. Figure 4 illustrates the structure of the metamodel. It brings together information from several sources:

- The message traces provided by the tool management console, e.g. the RabbitMQ tracing plug-in [6], for each message, it makes it possible to identify :
  - . The node through which it transits, as well as connection and host information.
  - . The exchange in which it was published or consumed, the queues to which it is routed or the queue from which it is consumed, and the associated routing keys.
  - . The user publishing or consuming the message.
  - . Its timestamp, its type (published / received) and its mode of delivery (persistent or not).
- The current configuration of the message broker, e.g. via the RabbitMQ REST management API[7].
- Events of creation and deletion resources, hosts, users and permission; creating and closing connections and channels and attempting to authenticate the user, e.g. using the RabbitMQ Event Exchange plug-in.[8].
- Contextual elements of the characteristics of the communicating applications provided by the business metadata.

We have analysed the 4 protocols and propose Table 2 which shows the differences between them and how we have integrated the concepts into the metamodel. The security concepts and the exchange concepts have also been presented in a previous article [18].

6.1 | System supervision queries and indicators

From the information collected, we can ensure the supervision of the system. Here is a subset of the queries and indicators that can be defined. These queries are needed for the company project but the metamodel is not limited to them.

- Business-level queries : Metadata may contain traces providing business-level information, i.e. information with an identifiable meaning, such as application identifiers, user identifiers, subjects and exchange formats. Similarly, the topics for which messages have been published when using a publish / subscribe messaging model can be specified. This information helps managers specify the context of interactions. For example, this allows one to take into account the identification of the user of the application that publishes or consumes messages.
- Message filtering : Messages can be filtered according to their characteristics such as identifier, timestamp, exchange, queues, editor, consumers, users, status, size, encoding or exchange format. This allows more precision when searching through the message traces. For

[6]https://www.rabbitmq.com/firehose.html
[7]https://pulse.mozilla.org/api/
[8]https://www.rabbitmq.com/event-exchange.html

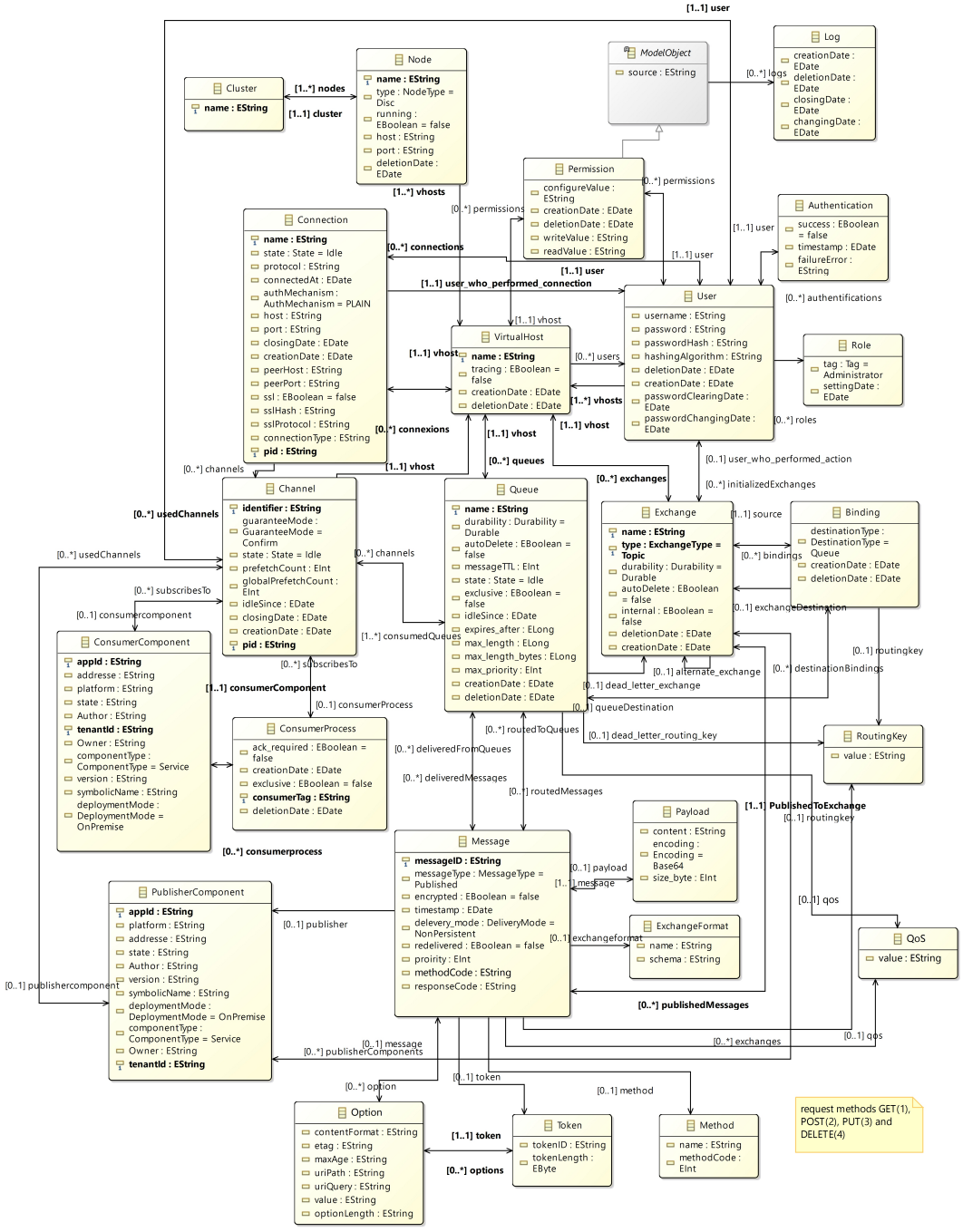

FIGURE 4 Pulse metamodel

example, filtering the messages that pass through a time interval between a producer and a consumer allows one to analyse the behavior of the interaction during a period when a failure occurs. We can also check whether a message is duplicated, redeposited or rejected.

- Security checks : Supervising the security of message exchanges can be improved by checking several elements:
  - User authentication timestamps and the success or failure of the authentication. This helps detect tampering attempts.
  - User and consumer permission on each resource to avoid potential data leaks.
  - The content of the payload is used to check whether the elements are encrypted or not.

TABLE 2 The Pulse terms associated with the protocols concepts

Pulse Terms	**AMQP**	**MQTT**	**KAFKA**	**CoAP**
Cluster	The collection and logical grouping of nodes that running the application and sharing users, virtual hosts, queues, exchanges, bindings, and runtime parameters	A distributed system that represents one logical MQTT broker, it consists of many different MQTT broker nodes that are connected over a network	Kafka cluster typically consists of multiple brokers to maintain load balance	Cluster is required to sync the CoAP servers to sync the common resource set
V-host	Designed for multi-tenancy and security reasons and used to separate permission of different users	MQTT connections use a single RabbitMQ host by default		CoAP connections use a single RabbitMQ host by default
User	A client application that pub/-sub messages to exchange/from queues	A client including publisher or consumer that pub/sub messages	Publisher and a group of consumers that act as a single logical unit	A client including publisher or consumer that pub/sub messages
Connection	A TCP network connection between users and the broker	It uses TCP to connect to the broker, keeping connection even no data exchanging	A framework for connecting Kafka with external systems such as databases, key-value stores and file systems	CoAP is a simplification of the HTTP protocol running on UDP network connection
Channel	A stream of communications which a lightweight connection to reduce the overhead of the operating system in establishing a TCP connection	A logical connection created by threads to communicate with broker	A type of Flume Channel that are the repositories where the events are staged on an agent	A logical connection created by threads to communicate with the broker
Binding	A virtual connection between exchange and queue, routing key can be included in binding	A virtual connection from RabbitMQ queues bound to the topic exchange		A virtual connection from a temporary RabbitMQ queues bound to the exchange
Queue	A named entity as the container of messages and delivers them to consumers	A named entity created for MQTT subscribers will have queue TTL in RabbitMQ		Subscription to a topic is implemented using a temporary RabbitMQ queue bound to that exchange
Routing Keys	A message attribute that the exchange may use when deciding how to route the message to queues	An address which topic exchanges route messages to queues based on specified wildcard (topic name)		Binding keys are compared for last value that was published with each routing key

- Evolution of architecture : The ability to track the creation and deletion of resources allows one to describe system configurations over time. In addition, based on the business information provided, we can identify communicating applications even if their technical identifiers have changed at the message broker level. We can therefore provide a more accurate visualization of the system components and their interactions, as well as their evolution over time.

- Interoperability indicators : the behavior of data exchange between applications gives an indication of the level of interoperability of their data. If, for example, a resource has been inactive for a certain period of time, this may indicate the uselessness or obsolescence of the interaction. It may also indicate a change of configuration (consumption on another channel) or a change of process at the application level. In Table 3, we describe how we can highlight some data interoperability issues through contextualized indicators or possible queries on the proposed metamodel. We also indicate potential causes and highlight existing correlations between interoperability problems.

6.2 | Evaluation of the Pulse metamodel

The validation of the Pulse metamodel was carried out according to the revised model evaluation framework proposed in Kitchenham *et al.* [50]. This validation involves qualitative aspect in terms of syntactic, semantic, pragmatic, test and value.

- Syntactic quality: it addresses the correctness of the syntax defined for the modeling. To this end, we performed a manual verification of the Pulse metamodel. The metamodel is developed using the UML language with the Eclipse modeling framework. The use of the Eclipse tool and the EMF framework ensures that the language is correctly used in the model, e.g. entities are represented by a rectangle with rounded corners, etc.

- Semantic quality: it addresses the traceability of the domain as well as feasible validity and completeness. The feasible validity inspection is aimed at checking that the model includes: definitions, detail and scope information. For this purpose, the different concepts of the monitoring are represented as a first-class entity. With regard to the feasible completeness, it concerns other domain features related to the pulse metamodel; problems associated with sensitivity analysis so as to identify unnecessary features consistency checking. This is illustrated in Table 2 that proves the metamodel reflects thus part of the domain. The first column of Table 2 takes up the different concepts of the Pulse metamodel. The column from two to five gives the meaning of the concepts for the four protocols covered by the scope of the model, respectively AMQP, MQTT, KAFKA, and CoAP. These concepts come from the datasheet of these protocols and linked to the Pulse concepts.

- Pragmatic quality: it addresses the structure and expressiveness of the model. We have built the metamodel representation considering the message as the core of the representation. As we aim to analyse the evolution of the messaging system all concepts are represented as first-class entities, with class names that are specific to the messaging system (Table 2). We assessed the level of comprehension achievable to enable comprehension including visualization, explanation, filtering evaluate comprehension presented in Section 7. Also, the metamodel is extensible to add new concepts.

- Test quality: it focuses on executability. We conducted out a validity test, using two case studies to assess consistency between our metamodel and case studies. For feasible test coverage, we have performed simulation studies based on importing data from the company related to input handling in order to assess the metamodel stability and sensibility (Section 7.1).

- Value: it addresses the practical usefulness of the metamodel and should enable one to use the model, including the design of an appropriate user interface for usability, user manuals and training. We proposed a prototype and our metamodel has proven that we are able to obtain generated figures representing supervised data through various case studies (Figure 6 and Figure 7).

6.3 | Application to Interoperability Analysis

Some approaches [51,52] provide the basic concepts for formalizing and assessing interoperability by indicating whether or not interoperability problems exist. Based on these concepts and in order to accurately locate interoperability problems among collaborative processes, Mallek et al. [53] define a set of interoperability requirements that must be checked in order to achieve interoperability. The requirements are structured in terms of compatibility, reversibility and interoperability properties defined for each interoperability problem (data, services, processes and activities). This formalization of interoperability requirements allows one to select the appropriate interoperability requirements to be verified.

We propose to focus on data requirements in order to assess data interoperability by analysing existing data exchanges between constituents. These requirements are related to elements that may vary or be modified during interactions. They must therefore be monitored. To do so, we inspect the behavior of data exchanges. This allows one to detect interoperability problems and find their potential causes.

Indicators and queries are used to detect and locate interoperability problems in order to maintain a good level of interoperability. To define a good level of interoperability, we rely on interoperability requirements, i.e. what must be undertaken and maintained when establishing interoperability interactions between communicating systems. Interoperability problems represent situations in which requirements are not verified. In addition to indicators that highlight these problems, we provide potential causes and highlight existing correlations between interoperability problems (by referencing other problems in the causes). If, for example, partners show no interaction at the data level over a certain period of time, this may indicate the uselessness or obsolescence of the interaction due to a change in the configuration (consumption on another channel) or a change in the process. It may also indicate an authorization problem or the invalidity of the exchange format.

Table 3 lists examples of how we can highlight data interoperability problems by providing indicators or possible queries that accurately define the context of the problem. For example, a data leak can be detected by the presence of unauthorized consumers or prevented by monitoring authentication attempts. The issues are difficult to identify in the context of distributed systems.

TABLE 3 Examples of interoperability requirements, interoperability problems, indicators and potential causes

Requirements	Interoperability problem	Operational indicator	Statistical indicator	Potential Causes
The data received are consistent with what required	Invalid exchange format	- Messages rejected on the relevant queues - Exchange format of the last message accepted on the queues	- Non-compliant message rate - Duplicate message rate	- Incompatible exchange format - Evolution of exchange format not communicated
Data is exchanged between partners	- Inability of consumers to receive more messages - Missing producer partner - Missing consumer partner	- Set of messages during the period concerned - Presence of the user linked to the Producer - Presence of Exchanges linked to the Producer	- Rate of messages transmitted - Rate of unconsumed messages - Rate of rejected messages - Rate of lost messages	- Problems related to the lack of acknowledgement of receipt - The consumer is not ready to receive - Disappearance of queues (deletion, failure) - Unable to restore the node containing the resources - Stopping the application - Permission problems
Partners have the necessary permissions to exchange data	Lack of necessary permissions to exchange data	- Authentication status - Connection status - Chain status - Presence of access permissions - Status of Vhost related permissions - user presence / user information	- authentication failure rate - logout rate - Authorisation change rate	- Deleted / non-existent users - Vhost deleted - User is not allowed to access - Limited permissions in Vhost - Permissions update not communicated - Authentication failed - Connection closed - Error in publishing or assigning consuming properties
The amount of data received is equal to the amount of data required	Consumption stop	- Presence of consumption process - Connection status - Channel status - Information of the last accepted message on the queues - Message accumulation in the relevant queues	- Rate of unconsumed messages - Rate of lost messages - Problems related to the lack of a consuming partner	- Problems related to lack of necessary permissions - Inadequate value of the Qos
Partners have an acceptable exchange rate	Low exchange rate	- Number of channels and components for a user - Number of queues in channel - Number of consumers in the channel	- Rate of lazy queues - Comparative rate of queues versus consumer by channel	- Sharing channels between threads - Use of the same channel as producer and consumer - Intensive resource creation - Few queues for many consumers - Missing consumer

6.4 | Implementation of the framework

We set up a prototype tool of the Pulse framework. It allows one to visualize some of the behaviors of the information system. This prototype aims at allowing one to understand the status of each message and provide indications which facilitate the analysis of interoperability problems.

The prototype is implemented on Moose.[9], an open-source data analysis platform [54,55], in which we have implemented the Pulse metamodel. The extension of Moose is a major asset to our implementation approach because it allows one to leverage and reuse existing tools.

The prototype shows the feasibility of our approach. Its implementation is composed of all the parts of the framework (Figure 5): the Pulse metamodel, the importers development, the historical record management thanks to Orion [49], the persistence layer, and some visualizations. The instantiation with the information retrieved from the information system follows a predefined sequence in order to build consistency in the history model (Figure 5) :

- Message Reader: it analyses the message trace log files and completes the data model. It is called only once at the beginning of the supervision to give an overview of the messages sent / received in the past.
- Management API Client: it uses the REST API if it exists. It is called the first time to fill in the missing information when reading the log files. It is then called on demand when new elements appear.
- Event Consumer : it continuously listens and captures events (events such as the definition of a new consumer, modification of user configurations, etc.) from the event queue and makes changes to the corresponding element in the data model.
- Messages Consumer : it constantly listens, captures new message traces and adds them to the data model as soon as they arrive.

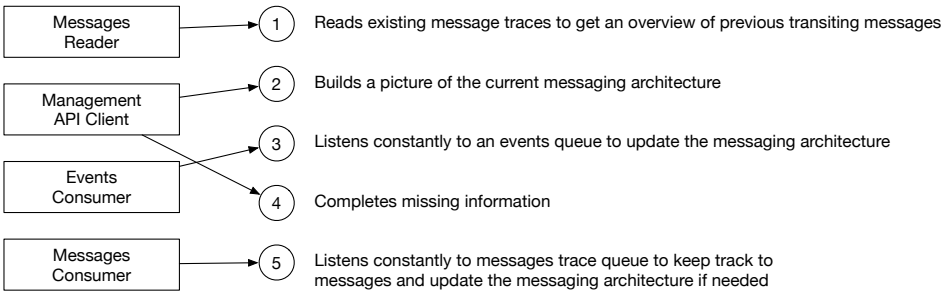

FIGURE 5 Information retrieval process with Pulse

The approach was already tested on data generated by an existing RabbitMQ implementation from Berger-Levrault. In a previous article [17], we provided analysis elements in response to some of the interoperability assessment needs expressed by the company requiring to have a clear visualization of the traces of messages in transit in each queue and their characteristics. This can be used for checking the status of messages during defined periods of time (Figure 6) and have a global vision of the information system architecture and message paths. This ensures the presence and activity of producers and consumers and the expected interactions between them (Figure 7).

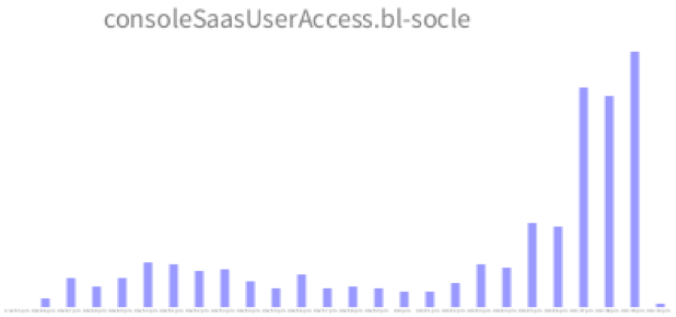

FIGURE 6 Number of messages per second in a queue

[9] https://modularmoose.org/

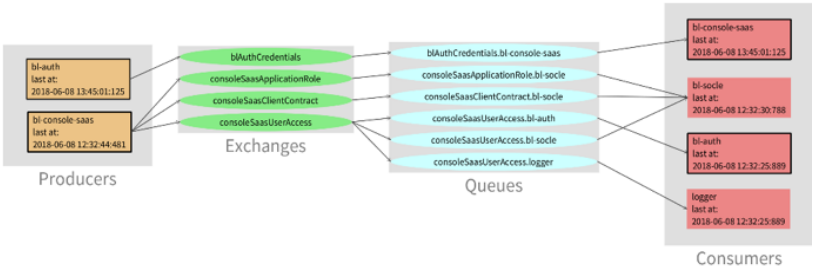

FIGURE 7 Structure of the analysed system

7 | CASE STUDY

We conducted two experiments. One deals with real data coming from a system on an AMQP base of the company Berger Levrault, on which we identified problems. We were able to improve the system following this analysis. The other deals with data from an MQTT simulator. The simulator makes several nodes communicate together. We imported the collected data to validate that the meta-model is functional with MQTT.

7.1 | Case study 1 : AMQP analysis

7.1.1 | Scenario

Here, we used a case of exiting interactions among Berger-Levrault applications to showcase some analysis services of the implemented prototype. Berger-Levrault provides its clients with a software as a service console (SAAS-Console) to ensure a secure access to its cloud deployed software via SSO (Single-Sign-On) mechanism. The SAAS-Console is also an administration console that allows clients to autonomously manage the accounts and access rights related to the software they use.

The SAAS-Console exchanges data with several applications (CRM, authentication modules and business applications). In this case, we considered here the ones that use RabbitMQ for data transmission: (i) BL-Auth, a separate authentication module, to which it sends user accounts rights and from which it receives credentials' updates, (ii) BL-Socle for the automatization of provisioning, to which it sends information regarding the packages of software to be deployed for the clients and their assigned user access accounts.

The interactions are set up with BL-MOM and the exchanged messages transit through a RabbitMQ node.

7.1.2 | Goals

In this case study, we provided analysis elements in response to some of behavior needs expressed by Berger-Levrault. We analysed only the events of the Event Consumer process.

- We ensured that the consumer connection and communication worked correctly. For that, we identified the failures and the differences between creation and deletion of artefacts. This need is related to the interoperability problem presented in Table 3, line 3.
- We managed to have a global vision of the behavior of the system, and detect unusual behaviors. This need is related to the interoperability problem presented in Table 3, line 5.

7.1.3 | Used Data

Berger-Levrault provided traces files from the Event Consumer used for the above explained data exchanges among the SAAS-Console, BL-Auth and BL-Socle. BL-MOM controls the messaging actions allowing one to have logs with business-level information. These log files are activated by elements that we want to trace on the broker node. The log file contains 738459 entries. We exploited only these log files without extracting information from other sources (e.g. REST API). For the sake of confidentially, we did not trace the payload of the messages which contained the transiting application data.

7.1.4 | Results

In the case study, we collected 738459 events (detailed in Table 4) for 5 days. The events concerned artefacts evolution during the lifecycle of the system. They concerned:

- the user authentication. It can be a successful authentication (2040 in the case study) or a failure authentication (detected 23 times). Related to the interoperability requirements and problems presented in Table3, it is related to the line 3, “Partners have the necessary permissions to exchange data”, and the indicator “Authentification status”.

- the connections. Two states were identified : the creation of the connection (2039 times), the closing of the connection (571901 times). These values could be used to analyse the interoperability requirements of lines 3 and 5 in the Table3.

- the channels. There were two states identified : the creation of a channel (79043 times), the closing of a channel (78865 times). These values could be used to analyse the interoperability requirements of lines 4 and 5 in the Table3.

- the consumers. There were two states identified : the creation of a consumer (1864 times), the deletion of a consumer (1711 times). These values could be used to analyse the interoperability requirements of lines 3 and 4 in Table3.

- the vhost. We identified 5 occurrences where the vhost goes down. Related to the interoperability requirements and problems presented in Table3, it is related to the line 2, “Data is exchanged between partners ”, and the indicator “Presence of link between a producer and a consumer”.

- the other artefacts. 968 events concerned other artefacts : the change of user tags, or password changes, or permission changes. Related to the interoperability requirements and problems presented in Table3, it is related to the line 3, “Partners have the necessary permissions to exchange data”, and the indicator “Authentification status”.

TABLE 4 Number of artefacts in the collected data

artefact	number
user authentication success	2040
user authentication failure	23
connection creation	2039
connection closure	571901
channel creation	79043
channel deletion	78865
consumer creation	1864
consumer deletion	1711
vhost down	5
other artefacts	968
Total	**738459**

Analyse of the number of artefacts :

We identified three interesting elements to be analysed. We first analysed the difference between created and closed artefacts. During the life cycle of the system, the number of created artefacts should be the same as the number of closed ones. In this case study, we started to analyse the system while it was already running. So, we cannot consider the whole life cycle. It means that the number of closed artefacts can be different from the number of created ones.

Secondly, the number of channels, and the number of consumers are slightly different. This is the phenomenon we have just explained and the company Berger-Levrault considered this difference as acceptable.

Thirdly, the number of closed connection was clearly higher than the created connections. The first idea was to consider that this number came from created connections before the monitoring process. But the number of creation (2039) was 280 times smaller than the number of closed connections. This difference raises the question of the origin of these closed connections. Regarding Table 3, the issue could be due to line 2, “Data is exchanged between partners”. An explanation was that consumers were waiting for data and then closed due to a timeout. This part should be investigated by the company, using other log files.

Analyse of the identified process :

We have identified a pattern of connection creation and message sending.

User authentication : it should succeed → connection creation → channel creation → consumer creation → consumer deletion → channel closing → connection closing.

This pattern is visible 2040 times. First of all, this pattern was presented to the company, which validated it : this process comes from the BL-MOM process structure. Then, this pattern was visible because the entities appeared successively and not interspersed with other events. Given the number of repetitions of this pattern, it was therefore quite easy to identify them. There is still work to do on identifying more complex or diffuse patterns.

Regarding Table 3 and the expertise of the company, we are able to identify authentication issues (line 3) related to problems of authentification failures.

Analyse of failures :

We detected three kinds of failures :

First, the vhost shut down 5 times during the period. In terms of Quality of Service, it is a problem. Just after the downtime, we identified that the users could reconnect with an authentication success. When a vhost shut down, all the connections, channels, authentications are lost. Even if, in the interoperability requirements and problems presented in Table3, this situation is related to line 2, “Data is exchanged between partners ”, it also has a strong impact on all other indicators because of an indirect impact on the loss of all running entities.

Secondly, there are 23 authentication failures. They are grouped in two periods:

- during the first period, there were 11 occurrences of authentication failures. Just after, there was a user creation coupled with the configuration tags and permission, followed by successful authentication.
- during the second period, there were 12 occurrences of authentication failures, which were followed by a password change and an authentication success.

Here, in the interoperability requirements and problems presented in Table3, the indicator “Authentification status” of the line 3, “Partners have the necessary permissions to exchange data”, is impacted and the problem, based on this indicator is identified.

Thirdly, we identified that sometimes, during short periods (between 1 and 2 minutes), creation and deletion of ephemeral channels occured : a channel was created and closed just after, without events detected during this period. For example, for one of these periods, there were more than 9000 events per minute, 4500 creation and 4500 closure channels. In the interoperability requirements and problems presented in Table3, this indicator “channel status” can impact the requirement on line 3, “Partners have the necessary permissions to exchange data” and the requirement on line 4, “The amount of data received is equal to the amount of data required”. Based on this analysis, the behavior will be investigated by the company.

7.2 | Case study 2 : MQTT messages importing

7.2.1 | Scenario and goal

Here, we used a simulator that simulates 15 IoT nodes which communicate together. We simulated air density sensors, humidity sensors, pressure sensors. The simulator stops when the message number created in the system is equal to 100 000. This home-made simulator allows us to generate large files of data using MQTT protocol [56].

In this case study, the goal is to evaluate the extensibility of the Pulse metamodel by the possibility to import MQTT messages in the same way as the AMQP messages.

7.2.2 | Used Data

The simulator generates trace files with all events during the time of the simulation (Figure 8). The generated file contains 100 000 entries. On each line, an event contains the payload, and all information provided by the system, for example the producer, the payload, the routing keys. The file size is 193 Mbits.

7.2.3 | Results

In the case study, we collected 100 000 events, as expected in the configuration of the simulator. We identified what we expected in the model analysis, shown in the left part of Figure 9 : 3 routing keys, 4 queues, 51 connections, 99895 messages, etc.

{"channel":1,"connection":"172.21.0.1:40218 ->
172.21.0.2:5672","exchange":"amq.topic","node":"rabbit@ec0aff581898","payload":"PE9 [...]
cz4=","properties":{},"queue":"none","routed_queues":["airdensity","all_messages"],"routing_keys":
["airdensity"],"timestamp":"2022-06-28
13:19:40:614","type":"published","user":"admin","vhost":"/"}

FIGURE 8 A part of the json file

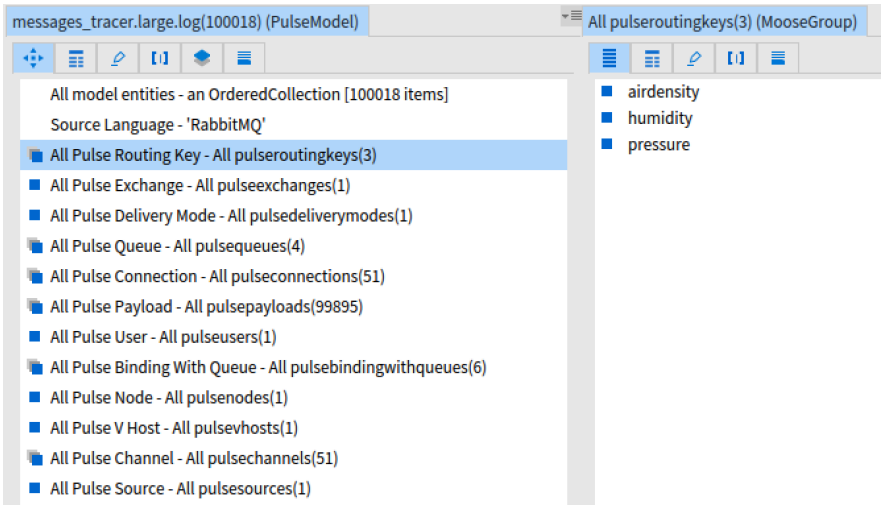

FIGURE 9 Pulse Model Information of MQTT log file

In this analysis, we can see that there are 99895 payloads. It is near 100 000. The difference probably comes from the similarity of multiple payloads. In the right part of Figure 9, we can see that there are 3 routing keys. These values are instantiated automatically from the content of the log file. There were 51 connections, which means that the 100 000 messages were sent during these 51 connections.

This case study shows that the Pulse metamodel is generic enough to integrate the MQTT protocol. It also validates that, when based on the log file, the Pulse framework discovered the architecture of the IoT system.

8 | DISCUSSION

In this section we discuss the limits and the future work of the proposed approach. We identify three elements to consider :

- The Pulse framework: Concerning the framework, we have validated the import thanks to parsers and we are able to do a static analysis. Indeed, we have not yet validated versioning management based on the Orion framework, in order to perform a dynamic analysis according to case study 1. Another point is that we have a working prototype based on the Pulse framework. The company wants a version of the evolved prototype with automatic analysis, especially by setting up configuration data, for example alert thresholds. Behind this tool, there is a research problem : how to adapt thresholds to the type of information system and information system activity ?

- The Pulse metamodel: We have proven that the model supports several protocols. This has been confirmed by two case studies on the AMQP protocol and MQTT protocol. The metamodel is extensible and can represent four communication protocols: AMQP, MQTT, Kafka, CoAP. The extensibility of the model is not yet validated for other communication protocols such as STOMP, ZigBee or ROS. One of the future works is to validate the extensibility with these protocols and control the impact on the metamodel.

- The evaluation of the interoperability problem: The model and the framework have been validated by two very different case studies. The first one is an industrial case study with real company data and the second one is done by simulation with prepared data. In the first case study, we used the indicators to manually identify the issues listed in Table3. In future work, we want to provide a recommendation system and, by extension, evaluate automatic problem identification.

9 | CONCLUSION

In this article, we have presented the means of supervising exchanges within a distributed information system, through a collaboration with Berger-Levrault company. The proposed framework addresses the services of message brokers to extract information on the behavior of the information system. We proposed a metamodel used to aggregate information gathered from several sources. The population of the metamodel provides contextual indicators to assess the interoperability needs related to the data and precisely locate interoperability problems.

We implemented the metamodel and analysis functions by extending the Moose metamodel and taking advantage of its extensibility. We developed a trace data importer and analyser to populate the metamodel. In response to some of Berger-Levrault's analysis needs, we used the data structured by the metamodel in order to build two data visualizations that represent multiple indicators.

In the achievements presented, we have shown that the supervision of distributed systems is essential to control and ensure the functioning of the system as a whole. We have also shown that semantic analyses can be used to optimize the handling of potential problems in a complex system.

In scientific literature, many locks for monitoring the dynamics of cyber-physical systems remain. These systems have the particularity of being heterogeneous, with hardware and software components, including so-called plug-and-play components. Cyber-physical systems ensure the linking of many sensors, as well as exchange large amounts of data while requiring low latency [29]. The difficulty today is to be certain that all the data produced will be consumed within a reasonable period of time. Several solutions exist at different levels: embedding control algorithms in the sensors, adapting the dimensioning of the network, using data orchestration software. These systems require adapted dynamic reconfiguration and monitoring mechanisms [57,58,59].

The increase in the volumes of data exchanged implies taking into account the variability in the frequency of data production. This requires that distributed architectures (both infrastructure and flow) are adaptable or self-adaptable [30] in order to reinforce the system's resistance to malfunctions and guarantee reliable interactions while avoiding congestion phenomena.

The availability of the resources used in the exchange architectures must be guaranteed to ensure system reliability. The variability of the types and frequencies of data exchanged can be managed by applying the principle of elasticity [60]. This principle makes it possible to automate the adaptation of the system according to the available components. This requires the definition of a set of exchange architecture configurations (optimized, stable, degraded, etc.) and mechanisms to switch from one configuration to another, in order to allow the resilience of the exchange system. The resilience capacity of these systems guarantees the functioning, but can hide the complexity of problems which are often difficult to solve in order to return to a stable state.

Acknowledgement : The authors would like to thank Catherine Batisse (Univ. Lyon 2) for proofreading this document.

References

1. O'Brien JA, Marakas GM, others . *Management information systems*. 6. McGraw-Hill Irwin . 2006.

2. Al-Mamary YH, Shamsuddin A, Hamid A, Aziati N. The role of different types of information systems in business organizations: A review. *International Journal of Research (IJR)* 2014; 1(7).

3. Shackelford R, McGettrick A, Sloan R, et al. Computing curricula 2005: The overview report. In: . 38. ACM. ; 2006: 456–457.

4. Int'l SG. Chaos Report. tech. report, Standish Group Int'l; : 2015.

5. Rasouli M, Ghazanfari H, Eshuis R. A process aware information system to support agility in relief operations. In: ; 2017.

6. Tarafdar M, Qrunfleh S. Agile supply chain strategy and supply chain performance: complementary roles of supply chain practices and information systems capability for agility. *International Journal of Production Research* 2017; 55(4): 925–938.

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

8. Geraci A, Katki F, McMonegal L, et al. *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press . 1991.

9. Gürdür D, Asplund F. A systematic review to merge discourses: Interoperability, integration and cyber-physical systems. *Journal of Industrial information integration* 2018; 9: 14–23.

10. Chen D, Daclin N, others . Framework for enterprise interoperability. In: ; 2006: 77–88.

11. Retaillé JP. *Refactoring des applications Java/J2EE*. Editions Eyrolles . 2011.

12. Panetto H, Zdravkovic M, Jardim-Goncalves R, Romero D, Cecil J, Mezgár I. New perspectives for the future interoperable enterprise systems. *Computers in Industry* 2016; 79: 47–63.

13. Li S, Da Xu L, Zhao S. The internet of things: a survey. *Information Systems Frontiers* 2015; 17(2): 243–259.

14. Kamble SS, Gunasekaran A, Parekh H, Joshi S. Modeling the internet of things adoption barriers in food retail supply chains. *Journal of Retailing and Consumer Services* 2019; 48: 154–168.

15. Avritzer A, Ferme V, Janes A, Russo B, Schulz H, Hoorn vA. A Quantitative Approach for the Assessment of Microservice Architecture Deployment Alternatives by Automated Performance Testing. In: Springer. ; 2018: 159–174.

16. Ceccarelli A, Bondavalli A, Froemel B, Hoeftberger O, Kopetz H. *Basic Concepts on Systems of Systems*: 1–39; Cham: Springer International Publishing . 2016

17. Amokrane N, Laval J, Lanco P, Derras M, Moala N. Analysis of Data Exchanges, Contribution to Data Interoperability Assessment. In: IEEE. ; 2018: 199–208.

18. Laval J, Amokrane N, Derras M, Moalla N. Analysis of Data Exchange among Heterogeneous IoT Systems. In: ; 2020; Tarbe, France.

19. Group CAW, others . Levels of information systems interoperability (LISI). *US DoD* 1998.

20. Clark T, Jones R. Organisational interoperability maturity model for C2. In: Citeseer. ; 1999.

21. Tolk A, Muguira JA. The levels of conceptual interoperability model. In: . 7. Citeseer. ; 2003: 1–11.

22. Guédria W, Chen D, Naudet Y. A maturity model for enterprise interoperability. In: Springer. ; 2009: 216–225.

23. Soria dIM, Alonso J, Orue-Echevarria L, Vergara M. Developing an enterprise collaboration maturity model: research challenges and future directions. In: IEEE. ; 2009: 1–8.

24. Kingston G, Fewell S, Richer W. An organisational interoperability agility model. tech. rep., Defence Science and Technology Organisation Canberra (Australia); : 2005.

25. Mallek S, Daclin N, Chapurlat V. Towards a conceptualisation of interoperability requirements. In: Springer. 2010 (pp. 439–448).

26. Verdecho MJ, Alfaro-Saiz JJ, Rodriguez R. Integrating business process interoperability into an inter-enterprise performance management system. *Proceedings of the 9th International Conference on Interoperability for Enterprise Systems and Applications (I-ESA) Berlin, Germany* 2018.

27. O'Connor RV, Elger P, Clarke PM. Continuous software engineering—A microservices architecture perspective. *Journal of Software: Evolution and Process* 2017; 29(11): e1866.

28. Johanssen JO, Kleebaum A, Paech B, Bruegge B. Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners. *Journal of Software: Evolution and Process* 2019; 31(5): e2169.

29. Buyya R, Dastjerdi AV. *Internet of Things: Principles and paradigms*. Elsevier . 2016.

30. Gascon-Samson J, Garcia FP, Kemme B, Kienzle J. Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud. In: IEEE. ; 2015: 486–496.

31. Da Xu L. Enterprise systems: state-of-the-art and future trends. *IEEE Transactions on Industrial Informatics* 2011; 7(4): 630–640.

32. Iacob ME, Jonkers H. A model-driven perspective on the rule-based specification and analysis of service-based applications. *Enterprise information systems* 2009; 3(3): 279–298.

33. Rubin VA, Mitsyuk AA, Lomazova IA, Aalst v. dWM. Process mining can be applied to software too!. In: ; 2014: 1–8.

34. Keith B, Vega V. Process mining applications in software engineering. In: Springer. ; 2016: 47–56.

35. Gürgen T, Tarhan A, Karagöz NA. An integrated infrastructure using process mining techniques for software process verification. In: IGI Global. 2018 (pp. 1503–1522).

36. Özdağoğlu G, Kavuncubaşı E. Monitoring the software bug-fixing process through the process mining approach. *Journal of Software: Evolution and Process* 2019; 31(7): e2162.

37. Aalst v. dWM. Process discovery from event data: Relating models and logs through abstractions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2018; 8(3): e1244.

38. Verenich I, Dumas M, La Rosa M, Nguyen H. Predicting process performance: A white-box approach based on process models. *Journal of Software: Evolution and Process* 2019; 31(6): e2170.

39. Gómez A, Iglesias-Urkia M, Urbieta A, Cabot J. A model-based approach for developing event-driven architectures with AsyncAPI. In: ; 2020: 121–131.

40. Michelson BM. Event-driven architecture overview. *Patricia Seybold Group* 2006; 2(12): 10–1571.

41. Maréchaux JL. Combining service-oriented architecture and event-driven architecture using an enterprise service bus. *IBM developer works* 2006; 12691275.

42. Theorin A, Bengtsson K, Provost J, et al. An event-driven manufacturing information system architecture for Industry 4.0. *International journal of production research* 2017; 55(5): 1297–1311.

43. Brand T, Giese H. Generic Adaptive Monitoring Based on Executed Architecture Runtime Model Queries and Events. In: ; 2019: 17-22

44. Dossot D. *RabbitMQ essentials*. Packt Publishing Ltd . 2014.

45. Vinoski S. Advanced message queuing protocol. *IEEE Internet Computing* 2006; 10(6): 87–89.

46. Standard O. MQTT version 3.1. 1. *URL http://docs. oasis-open. org/mqtt/mqtt/v3* 2014; 1.

47. Garg N. *Apache Kafka*. Packt Publishing Ltd . 2013.

48. Shelby Z, Hartke K, Bormann C. The Constrained Application Protocol (CoAP). RFC 7252, RFC Editor; : 2014.

49. Laval J, Denier S, Ducasse S, Falleri JR. Supporting Simultaneous Versions for Software Evolution Assessment. *Journal of Science of Computer Programming (SCP)* 2010.

50. Kitchenham BA, Pickard L, Linkman S, Jones P. A framework for evaluating a software bidding model. *Information and Software Technology* 2005; 47(11): 747–760.

51. Daclin N, Chen D, Vallespir B. Methodology for enterprise interoperability. *IFAC Proceedings Volumes* 2008; 41(2): 12873–12878.

52. Ford T, Colombi J, Graham S, Jacques D. The interoperability score. tech. rep., AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH; : 2007.

53. Mallek S, Daclin N, Chapurlat V. The application of interoperability requirement specification and verification to collaborative processes in industry. *Computers in industry* 2012; 63(7): 643–658.

54. Demeyer S, Tichelaar S, Ducasse S. FAMIX 2.1 — The FAMOOS Information Exchange Model. tech. rep., University of Bern; Bern, CH: 2001.

55. Ducasse S, Gîrba T, Kuhn A, Renggli L. Meta-Environment and Executable Meta-Language using Smalltalk: an Experience Report. *Journal of Software and Systems Modeling (SOSYM)* 2009; 8(1): 5–19. doi: 10.1007/s10270-008-0081-4

56. Riahi K, Kahn G, Dafflon B, Laval J. A Faulty IoT Network: Simulating Sensors and Perturbations. In: Springer. ; 2022: 87–97.

57. Vora O, Vora P, Vora U. Predictive Modeling for Infrastructure System Engineering. *Journal of Professional Issues in Engineering Education and Practice* 2003.

58. Dávid I, Ráth I, Varró D. Foundations for streaming model transformations by complex event processing. *Software & Systems Modeling* 2018; 17(1): 135–162.

59. Mohsin A, Janjua NK, Islam SM, Neto VVG. Modeling Approaches for System-of-Systems Dynamic Architecture: Overview, Taxonomy and Future Prospects. *architecture* 2019; 13: 15.

60. Al-Dhuraibi Y, Paraiso F, Djarallah N, Merle P. Elasticity in cloud computing: state of the art and research challenges. *IEEE Transactions on Services Computing* 2017; 11(2): 430–447.