Interoperability Mismatch Challenges in Heterogeneous SOA-based Systems

Cristina Paniagua∗†, Jens Eliasson∗ and Jerker Delsing∗
∗Dept. of Computer Science, Electrical and Space Engineering
Luleå University of Technology, Luleå, Sweden SE97187
†Email: cristina.paniagua@ltu.se

Abstract—In Industry 4.0, systems with a heterogeneous implementation, different architecture, protocols, and semantics have to be integrated into collaborative environments. The interoperability between them has become a major challenge in this new ecosystem of the industry, thereby generating several research questions about how to manage the information exchange and collaboration between systems with so vastly different properties.

This paper presents a detailed investigation of the different mismatch problems that can happen in the information exchange in heterogeneous SOA-based environments where the interoperability is crucial. The proposed solution in this paper is an Adapter System that can aid the generation of new service consumer interfaces at both compile-time and run-time. The interfaces are based on Service Contracts, thereby allowing heterogeneous systems to communicate and interchange services.

The proposed approach requires a new point of view in the service description field that can provide a holistic description of the information required for creating both run-time and compile-time consumer interfaces.

I. INTRODUCTION

The term Industry 4.0 was presented for the first time at the Hannover Fair [1] with the Industry 4.0 initiative, thereby initiating the fourth industrial revolution where devices and machines are connected to the Internet to communicate with other machines or humans. The achievement of this change in paradigm starts with the introduction of the Internet of Things (IoT) and Cyber-physical Systems (CPS) in industrial application scenarios [2].

The Internet of Things is composed of a large and heterogeneous set of devices, software systems, and networks in the quest of intelligent environments. During the last decades, a myriad of protocols and standards have been used to address the requirements of this new technological approach. The number of protocols and standards are growing as the number of new applications increase. Consequently, one of the major challenges today in the Industry 4.0 and the industrial IoT is the interoperability between systems with heterogeneous characteristics.

The collaboration between the different application domain silos and the promotion of the horizontal integration could generate a better and more complete functionality in the industry and open new possibilities but also challenges in the collaboration between systems.

Some of the IoT protocols and technologies commonly used in Service Oriented Architecture (SOA) are Representational State Transfer (REST) over HTTP, Constrained Application Protocol (CoAP) and OLE for Process Control Unified Architecture (OPC-UA). Each protocol has its own specialization and thus satisfies particular application requirements. In many cases, these protocols are designed as an isolated island where the communication between the islands is not considered.

The migration to a new collaborative horizon between communication protocols have been a hot discussion issue during the last years. Nowadays, there are commercial IoT platforms which support that interaction between communication protocols with different approaches. Some of the most common solutions are translation agents, gateways or cloud-based software buses. However, all these solutions do require integrated solutions routed through a central server, thus reducing flexibility and introducing security vulnerabilities as well as higher latencies. A new approach was presented by Derhamy et al. [3] in 2017 within the Arrowhead Framework as a SOA-based translator system. The approach uses an intermediate format instead of a protocol as a link between the target protocols, capturing all the specific information and improving scalability.

Nevertheless, the service interoperability issue is beyond just communication protocol mismatch. To achieve a holistic data exchange between heterogeneous systems, the semantics has to be taken into account, thus making existing solutions insufficient. In the literature, the semantic issue is defined and addressed in different manners. Liyanage et al. in [4] argued the use of ontologies to improve the interoperability in the field of health data, dividing the data into the semantic, syntactic and structural level and applying ontologies to classified the amount of big data across the healthcare ecosystem. Another perspective is presented by Jara et al. in [5] where analyses of the semantic interoperability in the IoT and argues that the Semantic Web of Things (SWoT) is the next step to achieve the integration of heterogeneous technologies. However, in the previous cases, the presented solutions still are in the development process, requiring human intervention and a considerable amount of input from domain experts or have scalability problems.

In addition, the notation used within the different semantics, understanding notation as a set of mappings and key value names, differs from one to another implementation without a general agreement, which makes the understanding between systems more complex.

This paper investigates mismatches in the different layers of
the communication between heterogeneous systems and the
necessity of achieving a functional solution, for this a new
approach is proposed. The new approach is based on the uti-
lization of the service contract, which specifies the interactions
between the consumer and the service provider, through the
creation of new solutions based on service orchestration and
data manipulation.

This paper is structured as follows: Section II presents
the background. Section III describes the problem in detail,
together with a few use cases. Section IV presents a new
approach to address the problem, Section V presents the
conclusions and Section VI provides suggestions for future work.

II. BACKGROUND

This section starts with a Service Oriented Architecture
overview and some of the most common protocols used in the
industry, followed by a short explanation about the concept
semantics and finally, a brief introduction of the Arrowed
Framework and its documentation.

A. Service Oriented Arquitecture

Service Oriented Architecture (SOA) is the use of software
interfaces called services to create distributed computing and
facilitate remote system interaction and data exchange. SOA is
based on principles such as the standardized service contract,
loose coupling, service abstraction and reusability among other
principles.

The service interface specifies the service operations, that is,
what the service does, the parameters that are passed into an
out the operation, and the protocols for how those capabilities
are used and provided [6].

The service contract is a concept introduced in SOA, it
is formed by the service interface, interface documents, the
service policies, and the service level agreement, including
Quality of service (QoS) and performance. The service con-
tract specifies all interactions between the service consumer
and service provider.

B. SOA protocols

Some of the most common SOA-enabled protocols used in the
industry are:

a) OPC UA: OPC-UA is a standard commonly used in the
process industry. OPC-UA allows a wide range of devices
to share information in a standardized way. OPC-UA is an
upgrade from the first OPC version, which has some serious
drawbacks such as being based on Windows-only technologies
such as DCOM. OPC-UA uses two different communication
protocols: Binary OPC over TCP allows high performance and
low overhead, or SOAP over HTTP(S) enables invocation of
Web Services.

OPC-UA also defines data formats and models and provides
mechanisms for alarms, security, information models, etc. A
comprehensive study of DPWS and OPC-UA was performed
by Candido et al. in [7]. Since OPC-UA runs over TCP, with
optional SOAP and HTTP(S) encoding, it is not suitable for
low-power devices that need to spend a great deal of their
time in sleep mode. The use of UDP uses fewer resources and
provides better a low-power operation.

b) HTTP: The Hypertext Transfer Protocol (HTTP) is
an application-level protocol for distributed, collaborative,
hypermedia information systems [8]. It is defined as a request-
response protocol in the client-server computing model. HTTP
is a TCP/IP based communication protocol, that is used to
deliver data on the World Wide Web. The three basic features
that make it a simple but powerful protocol are:

- Simple design makes implementations robust and inter-
operable.
- Media independent, any type of data can be sent as long
as the client and server understand it.
- Stateless, the server, and the client are aware of each other
only during a current request.

c) CoAP: In contrast with the others standards, CoAP [9]
was designed for the needs of constrained resource devices.
CoAP follows the client/server model and applies request-
response communication pattern. It is based on UDP and
provides an optional retry mechanism at the CoAP layer.

CoAP has a RESTful API with the GET, POST, PUT and
DELETE verbs supported with the addition of the OBSERVE
function. It is designed to interoperate with HTTP and the
RESTful world wide web at large though simple proxies.

C. The Arrowhead Framework

The Arrowhead Framework is an SOA-based framework
to support the creation of scalable cloud-based automation
systems [10]. One of its main challenges is to achieve the
interoperability between IoT devices and systems at the service
level, providing an exchange of information independently
of underlying protocols and semantic profiles. In Arrowhead
a system is considered a piece of software executed in a
device and what provide or consume services. The Arrowhead
Framework is based on the concept of System of systems.

The Arrowhead framework defines a properly structured
documentation to facilitate the integration of the systems. The
documentation allows documentation of SOA artifacts in a
common format [11]. It is divided in a three-level structure:
Systems of systems, system and service level. The main
aim of the document structure is to provide abstract and
implementation views of the System of systems, Systems, and
Services.

a) Systems of Systems level documentation: Systems of
Systems level documentation is divided into two documents,
the Systems of Systems Description (SoSD), which shows an
abstract view of a System of System, and the System of System
Desing Description (SoSDD), showing the implementation
view and its deployment views.

b) System level documentation: System level document-
tation is formed by two types of documents. The System
Description (SysD), which describes the systems as a black
box, documenting the functionality without describing its
internal implementation. And the System Design Description
(SysDD) that extends the black box description, showing internal details.

c) Service level documentation: Service level documentation consists of four documents: the Service Description (SD), the Interface Design Description (IDD), the Communication Profile (CP) and the Semantic Profile (SP). These four documents from the service contract in the Arrowhead framework.

- SD. Abstract view of the service. The document describes the main objectives and functionalities.
- IDD. Specific Interface implementation. The document points out the CP and the SP documents and provides information on how a service is implemented using the Communication and Semantics Profiles.
- CP. Contains all the information regarding the design protocol and the logical transportation of information.
- SP. Defines the encoding and the information semantics.

III. PROBLEM DEFINITION

In the new Industry 4.0 paradigm, the interoperability between different systems and devices has become a serious issue. The variety of commercial IoT platforms which support different communication protocols and semantic profiles make the communication between them a difficult task.

The idea of a renovated industrial environment more collaborative, where the machine to machine and machine to human synergy become more popular, opens new discussions about how to address the different obstacles in the communication and collaboration between them. Focusing on the machine to machine collaboration, one of the more challenging impediments are the numerous protocols, semantics, and standards currently used in the industry and that impede the exchange of services between systems.

In order to analyze the service interoperability issue, we have to break out the problem in different layers, as shown in Fig. 1. Hence, each layer has to be addressed and managed to achieve a complete solution. The convergence in a unique standard for each layer is considered a utopian idea, so the effort of the new investigations focus the collaborative between heterogeneous systems in all the different layers.

- Communication protocol. Includes the application layer protocols, e.g. HTTP, CoAP, STOMP, DDS, MQTT-SN.
- Encoding and serialization. Includes the syntactic format, e.g. XML, JSON, Protocol Buffers, Ion.
- Semantics. Includes standards that provide a meaning in a structured way, e.g. SenML, OMA SpecWorks, SensorML, AutomationML.
- Notation. Includes the mapping and the naming of the key values used, it is considered as part of the semantics. Some standards include a rigid notation while other permits some freedom or not include it.

<table>
<thead>
<tr>
<th>Format</th>
<th>Encoding</th>
<th>Semantic</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOTDB</td>
<td>JSON-LD</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>ASN.1</td>
<td>ISO/IEC 8825-1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IEEE 1451.4</td>
<td>–</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>RAML</td>
<td>JSON / XML</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Even when the technologies, shown in Table I, have a common format encoding, JSON or XML for example, the own semantic and notation use impede the communication between them.

There are several frameworks that include part of these layers in their specifications. Some of the most popular are the following. Thread, which is designed as a mesh of networks to securely connect hundreds of IoT devices. It is based on 6LoWPAN, IP Routing, and UDP. Another example is AllJoyn, a collaborative open source software framework that creates dynamic proximal networks using D-Bus message bus. Similar is the IoTivity, project hosted by The Linux Foundation and sponsored by the Open connectivity Foundation (OCF). Its goal is the creation of a new standard for wired and wireless devices will connect to each other and to the internet. Finally, IEEE P2413 is a draft Standard for an Architectural Framework for the Internet of Things, promoting cross-domain interaction and functional compatibility.

The Arrowhead framework is an SOA-based architecture which one of his objectives is the interoperability between systems beyond underlying protocols and semantics profiles, therefore, it is considered as the based for the analysis and future developments.

In the frame of SOA, to achieve a successful service interaction, the service consumer has to understand the data from the service provider. The consumption of a service requires the adequate consumer interface, traditionally this interface is the result of a previous agreement between the provider and the consumer based in the concept of service contract where is defined the characteristics of the service. The contract is accepted at the time of development and is used to implement the service consumer interface according to the service features to assure the correct performance of the systems within the local cloud. However, to achieve the collaboration environment systems with different contracts have to communicate and use services for which they do not have the adequate service consumer interface.

![Fig. 1. Service communication layers](image-url)
In this section, different cases where is described the interoperability problem are presented, in terms of lack of match between service contracts. For the analysis have been considered two arrowhead compliant systems, the system P is the provider of the service, and the system C is the consumer. Each system has a different service contract.

A. Case 1. Communication protocol mismatch

The first case under analysis is the mismatch between communication protocols, as it was previously described, some of the IoT protocols commonly used in SOA are HTTP, CoAP, and OPC-UA. Each protocol has its own scope and satisfies particular requirements. Most of the efforts in the field has focused on the interoperability at the network layer, with regard to the communication protocol.

There are some commercial solutions in the market to achieve the interoperability, such as translations agents, protocol gateways, such as the IPv6 Multi-protocol Gateway presented in [12] or the IoT Gateway system based on Zigbee and GPRS protocols proposed in [13], and adapters running embedded on the device [14]. In the Arrowhead framework, a support system called Translator was developed, which performs as a translator between protocols.

The difference between the service contracts in both systems is mainly in the communication profile. The main limitation of this method is the delay introduced in the communication, the message has to be sent to the translator and this has to translate and send to the other device, in addition, the use of a third system in each communication not only increase the communication time but also the risk of vulnerabilities and the security cost.

B. Case 2. Encoding and semantic mismatch

The second case of the analysis is when the semantic profile (SP) is different in both contracts. In the semantic profile is registered the data type, encoding, and semantics used in the service. In that sense, there are numerous combinations of the standards and encodings, precluding the understanding between systems. This situation does not have an easy solution due to the myriad of possibilities and the freedom of the developers in the design of the service.

Some of the obstacles in the consecution of the interoperability at this level are the difficulties for marking the necessary semantic information explicit, the requiring of human intelligence and judgement for solving mismatches, the documenting of legacy systems, the different points of view of different developers and the dynamic change of the meanings of names and values, such as Heiler explains in [15].

Most of the recent researches in the field of semantics are based on the use of ontologies. For example, Maree et al. propose in [16] an ontology merging framework that takes two domain-specific ontologies as input, finds semantic correspondences and produces a single merged ontology as output, claiming that the semantic heterogeneity problem can be solve merging two or more ontologies. In the literature it can be found ontologies particularly focused on IoT services, such as the one presented by Nambi et al. in [17] where it is shown a knowledge base for IoT, based on modeling contextual information as a result of merging several popular ontologies and their extension with relevant concepts in the field. Ngoc et al. in [18] go further the classical ontology concept, introducing the use of MAPE-k model from IBM as the fundamental reference model to implement autonomic systems and provide self-management capabilities.

However, the use of ontologies do not solve the entire problem, the ontologies describe a specific part of the knowledge, consequently, it is difficult to find a unique and accurate ontology that covers all the industrial semantic spectrum, even with the use of merging tools as the described before. In addition, these approaches are not totally developed or are still at a conceptual stage.

C. Case 3. Notation mismatch

The third case is focused on the deepest layer of the semantics, the mapping, notation, and naming of the message. The comprehension of the language is something intrinsic to the human being, however, the capabilities of understanding of the machines are limited to their implementation. Even when the communication profiles and semantic profiles are the same in two contracts, the freedom that the standards allow or even the possibility of not using standards in the description of the values becomes a big issue to communicate the systems.

This layer of the service understanding has not an official name, it is considered as a part of the semantics of the message. In this paper we want to differentiate this specific part from the term semantic, thus it is considered as the last and complex step in the quest of the interoperability in heterogeneous environments.

In the example shown in Fig. 5, all systems are using the same protocol, the same data application type (JSON) and both are describing the temperature in terms of value. But, despite the match in the characteristics of the service, the key values used in each system are different and this apparently small
differentiation in the semantic is the cause of the mismatch between the contracts, and consequently, the communication is not possible between them.

Code 5.a and 5.b have the same notation for the key value and the data type used, but the mapping of the message is different. Code 5.c, on the other hand, has the same mapping than the 5.a, but the notation used in the key value is different. In both cases, the differentiation between the messages are slightly but enough to make an error in the communication and make the interoperability impossible.

D. Case 4. Combination of several different mismatches

The last case is the combination of the previous cases, the previous cases may happen together or in any combination of them. The more heterogeneous is the system the more complex is the solution.

IV. PROPOSAL

The complexity of the countless combination of structures, protocols, data types, encodings, and semantics, standards make the idea of the convergence into one universal solution a utopic idea far from the current industrial reality.

The service consumer interface is the point through which consumer interact with the provider. The interface defines the style and details of the interactions, unlike the implementation, which defines how a particular provider offers its capabilities. The distinction between of these two terms is critical to understand the operation of the service.

The proposed solution considers the interface as the main point. With the appropriate consumer interface, it would be possible to consume the provider service and achieve the interoperability between the different systems.

However, to modify the software of the consumer service interface to adapt it to the contract of the provider’s service is considered as a bad practice. A change can affect the functionalities of the system and the increase the errors. The proposed alternative is to add an intermediate instance that acts as a link between both interfaces; this instance could be considered as a new dynamic consumer interface capable of manipulating and adapt the data for been use for the consumer.

This paper proposes a new system called Adapter System in charge of creating instances capable of manipulating the data interchange between the producer and the consumer. The Adapter uses the information stored in the Service Contract of each System. Therefore, the correct management of the metadata of the service, as well as, the service contract documentation becomes fundamental to resolve the interoperability issue.

Formats such as Web Application Description Language (WADL) or Web Services Description Language (WSDL) are designed to provide a machine-readable description of web applications and services. However, they are restricted to a particular protocol and standard limiting their use and they are not sufficient for the description of the service contract in its totality, and therefore, for the generation of consumer interfaces. This paper proposes the evolution of a new format capable to adapt to different protocols and standards and describe the information necessary to generate the consumer interface in compile-time and run-time. This format is called Contract Description Language (CDL). The CDL is an XML-based interface definition language that is used for describing service contracts and is in its first steps of development.

The principal difference with other formats is the detailed description of the encoding, compression, security, QoS, semantic, and notation of responses and requests, as well as, a mapping format that separates the structure of the message from the used syntactic format. The proposed approach allows to compare two different service contracts and analyze the mismatch between them and thereby present a solution that can enable communication between two mismatching services.

To test the applicability of the proposed approach, a basic implementation of the Adapter system in a Java program was developed. The Adapter is capable of reading two CDLs from different systems and analyzing the mismatches, follow by proposing a solution, if needed, Fig. 7.
The possible solutions depend on the mismatches between both service contracts. If there is not any mismatch between them, the systems are compatible and no action is required. In case of mismatch in the communication protocol or encoding the solution is to send a new orchestration rule that calls the translator system. On the other hand, when the problem is related to the semantic and notation a possible solution can be the data manipulation. However, there are cases where none of this solutions can be applied or they are not enough to solve the mismatch, in this situation the adapter does not generate any action.

V. CONCLUSION AND RESULTS

The new technological approach that started with Industry 4.0 is today facing new interoperability issues. Today, there are a number of solutions available that partially solve some of the issues. However, there is not yet a holistic solution to achieve a complete communication and understanding between systems with different service contracts.

To achieve a collaborative environment between heterogeneous systems the different layers of the service contract has to be taken into account. The proposed characterization divides the service into the communication protocol, the encoding, the semantic and the notation of service’s messages.

This paper provides an investigation of the interoperability issues in heterogeneous SOA-based environments and proposes a new approach where the generation of run-time and compile-time dynamic consumer interfaces enables communication between heterogeneous systems.

The first step in the creation of the new interfaces is the management of the service metadata. This approach requires a new method for service description that can capture all essential information required for creating consumer interfaces. The solution proposed is focused on the definition and implementation of a machine-readable description of the service contract, the Contract Description Language (CDL). Results from this paper is a major step towards efficient collaborative automation.

VI. FUTURE WORK

The future work will be focused on the consecution of the interoperability of the different layers through the automatic generation of dynamic consumer interfaces, and its validation in real industrial use cases. That involves the investigation of dynamic re-orchestration rules and data manipulation at runtime. The CDL specification will be elaborated and extended to enable more details of mismatch. The QoS and the security necessities need to be defined and included in future solutions. The results will be integrated into the future implementations of the Arrowhead Adapter System.

ACKNOWLEDGMENT

The authors would like to thank the European Commission and the Productive 4.0 project (EU ARTEMIS JU grant agreement no. 737459) for funding.

REFERENCES