

Object-Oriented Internet Cloud Interoperability

Mariusz Postół¹[0000–0002–9669–0565] and Piotr Szymczak¹[0000–0001–6790–3878]

Institute of Information Technology, Lodz University of Technology, Łódź, Poland
`mailto:mariusz.postol@p.lodz.pl`

Abstract. Optimization of the industrial processes requires further research on the integration of machine-centric systems with human-centric cloud-based services in the context of new emerging disciplines, namely Industry 4.0 and Industrial Internet of Things. This research aims at working out a new generic architecture and deployment scenario applicable to this integration. The reactive interoperability relationship of the interconnected nodes is proposed to deal with the network traffic propagation asymmetry or assets' mobility. Described solution based on the OPC Unified Architecture international standard relaxes issues related to the real-time multi-vendor environment. The discussion addressing the generic architecture concludes that the embedded gateway software part best suits all requirements. To promote separation of concerns and reusability, the proposed architecture of the embedded gateway has been implemented as a composable part of a selected OPC UA PubSub framework.

The proposals are backed by proof of concept reference implementations confirming the possibility to integrate selected cloud services with the cyber-physical system interconnected as one whole atop of the OPC UA by applying the proposed architecture and deployment scenario. It is contrary to interconnecting cloud services with a selected OPC UA server limiting the PubSub role to data export only.

Keywords: Industry 4.0 · Internet of Things · Object-Oriented Internet · Cloud Computing · Industrial communication · Reactive Networking (RxNetworking) · Machine to Machine Communication · OPC Unified Architecture · Azure

1 Introduction

All the time, the Information and Communication Technology is providing society with a vast variety of new distributed applications aimed at micro and macro optimization of the industrial processes. Obviously, the design foundation of this kind of application has to focus primarily on communication technologies. Based on the role humans take while using these applications they can be grouped as follows:

- **human-centric** - information origin or ultimate information destination is an operator,

- **machine-centric** - information creation, consumption, networking, and processing are achieved entirely without human interaction.

A typical **human-centric** approach is a web-service supporting, for example, a web user interface (UI) to monitor conditions and manage millions of devices and their data in a typical cloud-based IoT approach. In this case, it is characteristic that any uncertainty and necessity to make a decision can be relaxed by human interaction. Coordination of robots behavior in a work-cell (automation islands) is a **machine-centric** example. In this case, any human interaction must be recognized as impractical or even impossible. This interconnection scenario requires the machine to machine communication (M2M) [8, 10, 13, 15, 18] demanding multi-vendor devices integration.

From the M2M communication concept, a broader idea of a smart factory can be derived. In this M2M deployment approach, the mentioned robots are only executive assets of an integrated supervisory control system responsible for macro optimization of an industrial process composed as one whole. Deployment of the smart factory concept requires a hybrid solution and interconnection of the mentioned above heterogeneous environments. This approach is called the fourth industrial revolution and was coined as Industry 4.0. It is worth stressing that machines - or more general assets - interconnection is not enough, and additionally, assets interoperability has to be expected for the deployment of this concept. In this case, multi-vendor integration makes communication standardization especially important, namely, it is required that the payload of the message is standardized to be factored on the data-gathering site and consumed on the ultimate destination site.

Highly-distributed solutions used to control real-time process aggregating islands of automation (e.g. virtual power plants producing renewable energy) additionally must leverage public communication infrastructure, namely the Internet. Internet is a demanding environment for highly distributed process control applications designed atop the M2M communication paradigm because

- it is a globally shareable and can be also used by malicious users
- it offers only non-deterministic communication making integration of islands of automation designed against the real-time requirements a difficult task

Today both obstacles can be overcome, and as examples, we have bank account remote control and voice over IP in daily use. The first application must be fine-tuned in the context of data security, and the second is very sensitive in regard to time constraints. Similar approaches could be applied to adopt the well known in process control industry concepts:

- Human Machine Interface (HMI)
- Supervisory Control and Data Acquisition (SCADA)
- Distributed Control Systems (DCS)

A detailed examination of these solutions is far beyond the scope of this article. It is only worth stressing that, by design, all of them are designed on the

foundation of interactive communication. Interactive communication is based on a data polling foundation. In this case, the application must follow the interactive behavioral model, because it actively polls the data source for more information by pulling data from a sequence that represents the process state in time. The application is active in the data retrieval process - it controls the pace of the retrieval by sending the requests at its convenience. Such a polling pattern is similar to visiting the books shop and checking out a book. After you are done with the book, you pay another visit to check out another one. If the book is not available you must wait, but you may read what you selected. The client/server archetype is well suited for the mentioned above applications.

After dynamically attaching a new island of automation the control application (responsible for the data pulling) must be reconfigured for this interoperability scenario. In other words, in this case, the interactive communication relationship cannot be directly applied because the control application must be informed on how to pull data from a new source. As a result, a plug and produce scenario [11] cannot be seamlessly applied. A similar drawback must be overcome if for security reasons suitable protection methods have been applied to make network traffic propagation asymmetric. It is accomplished using intermediary devices, for example, firewalls, to enforce traffic selective availability based on predetermined security rules against unauthorized access.

Going further, we shall assume that islands of automation are mobile, e.g. autonomous cars passing a supervisory controlled service area. In this case, the behavior of the interconnected assets is particularly important concerning the environment in which they must interact. This way we have entered the Internet of Things domain of Internet-based applications.

If we must bother with the network traffic propagation asymmetry or mobility of the asset network attachment-points the reactive relationship could relax the problems encountered while the interactive approach is applied. In this case, the sessionless publisher-subscriber communication relationship is a typical pattern to implement the abstract reactive interoperability paradigm. The sessionless relationship is a message distribution scenario where senders of messages, called publishers, do not send them directly to specific receivers, called subscribers, but instead, categorize the published messages into topics without knowledge about which subscribers if any, there may be. Similarly, subscribers express interest in one or more topics and only receive messages that are of interest, without knowledge about which publishers, if any, there are. In this scenario, the publishers and subscribers are loosely coupled i.e they are decoupled in time, space and synchronization [4].

If the **machine-centric** interoperability - making up islands of automation - must be monitored and/or controlled by a supervisory system cloud computing concept may be recognized as a beneficial solution to replace or expand the mentioned above applications, i.e. HMI, SCADA, DCS, etc. Cloud computing is a method to provide a requested functionality as a set of services. There are many examples that cloud computing is useful to reduce costs and increase robustness. It is also valuable in case the process data must be exposed to many stakeholders.

Following this idea and offering control systems as a service, there is required a mechanism created on the service concept and supporting abstraction and virtualization - two main pillars of the cloud computing paradigm. In the cloud computing concept, virtualization is recognized as the possibility to share the services by many users, and abstraction hides implementation details.

Deployment of the hybrid solution providing interoperability of the **machine - centric** Cyber-Physical System (CPS) and **human - centric** cloud-based front-end can be implemented applying the following scenarios: **direct** or **gateway** based interconnection (Sect. 2.1).

By design, the **direct** approach requires that the cloud has to be compliant with the interoperability standard the CPS is built upon - it becomes a consistent part of the CPS. Data models, roles, and responsibility differences of both solutions make this approach impractical or even impossible to be applied in typical cases. A more detailed description is covered by the Sect. 2.

This article addresses further research on the integration of the multi-vendor **machine-centric** CPS designed atop of M2M communication and emerging cloud computing as a **human-centric** front-end in the context of the Industry 4.0 (I4.0) and Industrial Internet of Things (IIoT) disciplines. For this integration, a new architecture is proposed to support the reactive relationship of communicating parties. To support the multi-vendor environment OPC Unified Architecture [6] interoperability standard has been selected. Prototyping addresses Microsoft Azure Cloud as an example. The proposals are backed by proof of concept reference implementations - the outcome has been just published on GitHub as the open-source (MIT licensed) [16]. The proposed solutions have been harmonized with the more general concept called the Object-Oriented Internet (OOI) [16].

The main goal of this article is to provide proof that

- reactive interoperability M2M communication based on the OPC UA standard can be implemented as a powerful standalone library without dependency on the Client/Server session-oriented archetype
- Azure interoperability can be implemented as an external part employing out-of-band communication without dependency on the OPC UA implementation
- the proposed generic architecture allows that the gateway functionality is implemented as composable part at run-time part - no programming required

The remainder of this paper is structured as follows. Sect. 2 presents the proposed open and reusable software model. It promotes a reactive interoperability pattern and a generic approach to establishing interoperability-context. A reference implementation of this archetype is described in Sect. 3. The most important findings and future work are summarized in Sect. 4.

2 Sensors to Cloud Field Level Connectivity

2.1 Architecture

As it was explained in Sect. 1, to follow the Industry 4.0 concept a hybrid environment integrating reactive Machine to Machine interconnection and the interactive web-based user interface is required (Sect. 1). The main challenge of the solution in concern is to design a generic but reusable architecture that addresses interoperability of these diverse interconnection scenarios ruled by different requirements, namely:

- **machine-centric** machine to machine real-time mobile interoperability
- **human-centric** cloud-based front-end

Interconnection of the reactive **machine-centric** and interactive **human-centric** environments can be implemented by applying one of the following scenarios:

- **direct interconnection** (tightly coupled scenario) - cloud-based dedicated communication services are engaged to attach it to the CPS making up a consistent M2M communication network using a common protocol stack
- **gateway based interconnection** (loosely coupled scenario) - native build-in communication services allows attaching the cloud to the CPS using an out-of-bound protocol stack

In the solution in concern, the interconnection of assets is not enough hence their interoperability is expected. In this case, using the same communication stack must be recognized as only a necessary condition. To support interoperability common data understanding is required. By design, the direct approach requires that the cloud has to be compliant with the interoperability standard the CPS uses. As a result, it becomes a consistent communication node of the CPS. Additionally, to meet this requirement the cloud and CPS have to establish directly the same:

- semantic-context
- security-context

The possibility to establish a common semantic-context in the multi-vendor environment makes communication standardization especially important. In this case, it is required that the encoding of the payload exchanged over the network (Data Transfer Object - DTO) is standardized so that the appropriate messages can be factored on the data-gathering site and consumed on the ultimate destination data processing sites. Security between the data origin and ultimate data destination refers to the protection of messages (security-context) against malicious users. It is required that communicating parties are using the same cyber-security measures.

The decision to follow the **direct interconnection** scenario must be derived from an analysis of the capabilities of available services in concern. However, for the development strategy of this type of solutions this analysis can be done partially taking into account two features that can be considered invariable:

- by design the cloud-based services must be virtual - they are used to handle many solutions at the same time
- by design the M2M communication is usually constrained by the real-time requirements

The virtualization of cloud services means that they must be very flexible to handle the attachment of new assets proactively (acting in advance) at run time. As a result, by design, the cloud services must be responsible to register and authenticate devices by exposing endpoints in the public network to allow the device to access a provisioning cloud service. It requires that a session over the Internet has to be established by the data holding asset at a preparation step.

To meet the requirements of real-time distributed control the CPS may use protocols applicable only to local computer networks (e.g. multicast IP, Ethernet, TSN¹, etc.). Because the cloud services support only protocols handling interconnection over the Internet the interaction with the cloud requires remote agents, i.e. agents attached locally to the M2M network and implemented by applying one of the following archetypes:

- **edge device** - remote cloud agent acting as an intermediary for nodes of the CPS
- **field level gateway** - a dedicated custom agent acting as an intermediary for nodes of the CPS
- *Embedded Gateway* - a software part composed into a selected node of the *Cyber-physical network* (Fig. 1)

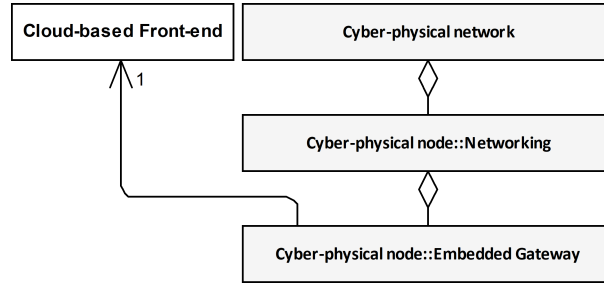


Fig. 1. Domain model of a generic interconnection concept

Edge device connects directly to the cloud services but acts as an intermediary for other devices called leaf devices. Additionally, it allows the selection of initial data processing and execution of them using local resources. The **edge device** may be located close to the leaf devices and attached to the *Cyber-physical network* using protocols applicable only to local computer networks. In

¹ Time-Sensitive Networking (TSN) Task Group <https://1.ieee802.org/tsn/>

this scenario, it is possible to use a custom protocol stack to get connected to the **edge device** with the cloud and helps to save the bandwidth thanks to sending only the results of local processing. In this approach, the **edge device** is part of cloud vendor products and cannot be recognized as a generic solution that can be used to connect to other clouds at the same time.

The **field level gateway** is also build atop of the middleware concept [17]. The only difference compared with the **edge device** is a necessity to use officially supported by the cloud vendor services to get connected. In this scenario, the process data simultaneously may be transferred to many clouds provided that the gateway offers this functionality.

Unlike the above-described solutions, the *Embedded Gateway* is not derived from the middleware concept. A generic domain model for this interconnection is presented in the Fig. 1. Promoting separation of concern design principle, the gateway functionality should be implemented as a self-contained software part embedded in the *Networking* service of the *Cyber-physical node*. The main functionality of this component is to transfer selected data between *Cyber-physical network* using *Networking* services of an existing *Cyber-physical node* and *Cloud-based front-end* using officially supported by the cloud vendor interconnection services.

The *Embedded Gateway* archetype relaxes most of the issues described above: *Cyber-physical network* real-time behavior, data encoding incompatibility, security - context differences to name only a few. The main goal of this article is to provide proof that the *Embedded Gateway* archetype implementation is possible based on a generic architecture that can be used as a foundation for the integration of the heterogeneous environments in concern. The proposed implementation is designed for a selected interoperability standard and cloud product.

To comply with the Industry 4.0 communication criterion it is required that any product must be addressable over the network via TCP/UDP or IP and has to support the OPC UA Information Model. As a result, any product being advertised as Industry 4.0 enabled must be OPC UA-capable somehow. To support the multi-vendor environment OPC Unified Architecture interoperability standard has been selected. OPC UA supports the following two patterns to be used to transfer data between communicating parties:

- **session-oriented**: requires a session that has to be established before any data can be sent between sender and receiver
- **sessionless-oriented**: the sender may start sending messages (called packets or datagrams) to the destination without any preceding handshake procedure

Using the session-oriented communication pattern it is difficult or even impossible to gather and process mobile data (Sec. 1), which is one of the Internet of Things paradigms. OPC UA Part 14 PubSub [2, 14] offers the sessionless approach as an additional option to session-based client-server interoperability relationship and is a consistent part of the OPC UA specifications suit. As the result, it can be recognized as the IoT ready technology.

The presented proposals in the article are backed by proof of concept reference implementations [16]. For this study, prototyping addresses Microsoft Azure cloud products. There are many reasons for selecting Azure to accomplish the cloud-based front-end of Cyber-Physical System (CPS). Azure offers Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) capabilities. As a result, the platform can be used not only as a cloud-based front-end for CPS. By design, the Azure services are compliant with Security Development Lifecycle (SDL) an industry-leading security process. It is also compliant with the new international standard for cloud privacy, namely ISO 27018. Solutions hosted on Azure are scaled up to millions of users without any additional coding. For the development of the CPS front-end, it is essential that Azure provides very efficient storage services usefully for the purpose of real-time process data archival. Azure provides a vast variety of hybrid connections including but not limited to virtual private networks (VPNs), caches, content delivery networks (CDNs), ExpressRoute, and IoT dedicated services that can be directly used to implement cloud-based front-end for CPS. Because it is also integrated with other Microsoft tools like Office 365, Outlook, and SharePoint using Azure allows preserving investment and exporting process data to the mentioned tools. Azure also offers services supporting analytics and intelligence capabilities for further improving business processes and decision making. It is the only cloud platform that offers Blockchain as a Service (BaaS), Machine Learning, Bots, and Cognitive APIs capabilities.

Azure aids Internet protocols and open standards such as JSON, XML, SOAP, REST, MQTT, AMQP, and HTTP. A software development kits for C#, Java, PHP, and Ruby are available for custom applications. Azure provides services supporting data exchange over the OPC UA, but they don't support PubSub compliant with the OPC UA Part-14. Connectivity services on the network use JSON-based Data Transfer Object encoded based on schema derived from the solution metadata.

A more detailed description of the selected Azure features in the context of the application in concern is covered by the Sec. 2.2.

Based on the sessionless and session-oriented communication patterns examination against the IoT requirements [13] it could be concluded that the connectionless pattern better suites issues related to the assets mobility and traffic asymmetry that is characteristic for the application domains in concern. Additionally, to promote interoperability and address the demands of the M2M communication in the context of a multi-vendor environment the prototyping should use a framework that must be compliant with the OPC UA Part 14 PubSub specification. According to proposed generic architecture (Fig. 1) to implement the *Embedded Gateway* as a composable part of the *Cyber-physical node* a library implementing *Networking* functionality in compliance with mentioned above specification is a starting point for further development. Additionally, it must be assumed that the library used to deploy *Embedded Gateway* support dependency injection and be capable to compose an external part supporting Azure/PubSub gateway functionality. The composition process must be avail-

able without modification of the core code of an existing library. As a result, the prototyping is to be limited to implementation of the *Embedded Gateway* software part only.

A library that meets all these requirements has been implemented consistently with the Object-Oriented Internet (OOI) paradigm [16] worked out in an open-source project ². The paper [13] covers the description of a reference application program implementation proving that it is possible to design universal architecture targeting reactive interoperability as a consistent part of the OOI concept compliant with the OPC UA PubSub international standard. According to the presented implementation and evaluation, using the dependency injection and late binding, the application program can be seamlessly adapted to the production environment and scales well. This approach also improves flexibility and adaptability of the existing solutions against any modification of the production environment, including but not limited to the selected interoperability standard change.

Sect. 2.3 provides a more detailed description of this library and an external part deployment process that is to be used to implement new functionality supporting *Embedded Gateway*.

The following subsections cover the description of the current state of technologies concerning Azure cloud-based IoT enabler and OPC UA PubSub. Sect. 2.2 analyzes data presentation user interface, available native communication services, and data/metadata model offered by Microsoft Azure. The discussion covered by this section is the foundation for selecting services utilized to expose process data and suitable protocol stack to support interconnection. In Sect. 2.3 the discussion focuses on the generic architecture that is to be used as a foundation for further decisions addressing the systematic design of the interoperability of the CPS and cloud-based front-end.

2.2 Azure Main Technology Features

Deployment of the hybrid solution providing interoperability of the **machine-centric** Cyber-Physical Systems (CPS) designed atop of M2M reactive communication and emerging cloud computing as a **human-centric** front-end requires decisions addressing the selection of the services supporting web user interface capable to expose process data. In this context, the service is any autonomous (with its own identity) software component or module that is interfacing with selected CPS for data collection, analysis, and also remote control. Microsoft Azure is a cloud-based product. It offers a vast variety of services. This virtual environment handles an unlimited number of users and devices organized using a solution container concept. The solution aggregates users, devices, services, and required additional resources scoping on a selected scenario. It also serves as a context that provides a scope to the identifiers (the names of devices, users, process data entities, etc) inside it. Solutions are used to organize deployment entities into logical groups and prevent identity collisions.

² <https://github.com/mpostol/OPC-UA-OOI>

The **IoT Central** service provides a user interface of process data visualization. To make this interface meaningful, metadata called device template is used to describe devices.

Following the assumption that interconnection between the CPS and cloud services is designed based on the gateway concept, a middleware must be considered as a coupler. It must be interconnected with the CPS using an in-band protocol adhering to communications requirements (i.e. protocol profile, data encoding, time relationships, etc.) governing communication of the parts making it up. At the same time, it must support back-and-forth data transfer to the cloud using out-of-band native for the cloud services. The transfer process requires data conversion from source to destination encoding. The **IoT Hub** is a service hosted in the cloud that supports **IoT Central** services providing a robust messaging solution - it acts as a central message hub for bi-directional communication [3]. This communication is transparent, i.e. it is not data types aware allowing any devices to exchange any kind of data. This service is responsible to manage the devices' identity and it offers the following protocol stacks: AMQP, MQTT, and HTTPS.

Before process data can be exposed using a web user interface the data source must be associated with an appropriate solution and validated to make sure that the security rules are not violated. It is hard to assume that the security rules governing the CPS may also apply to the cloud-based services. In the gateway scenario, they can be mapped on each other or be entirely independent. The *IoT Hub Device Provisioning Service* (DPS) is a helper service for *IoT Hub* that enables devices' connection process management, upon device providing valid identity attestation it assigns the device to an appropriate *IoT Hub* instance and returns to the device connection parameters, which allow direct connection with given *IoT Hub* service. The device proceeds to use the same attestation in *IoT Hub* connection and based on it, is granted authorization to selected operations including but not limited to data transfer and updating the user interface.

It is worth stressing that interaction of the offered by the Azure services can be configured flexibly, and as a result, the presented above selection of services must be recognized as an example only. The **IoT Central** can be also seamlessly integrated with other services as needed. The following services could also be considered to build a cloud-based automation solution:

- *Industrial IoT* - discovering OPC UA enabled servers in a factory network and register them in Azure *IoT Hub* implemented using *IoT Edge*
- *Digital Twins* - managing the graph of digital twins, which are to represent some real-world process or entity

Industrial IoT promotes OPC UA client/server archetype used to achieve direct and interactive interoperability implemented using *IoT Edge* services that allow extracting initial data processing to local premises based on the edge concept. *Digital Twins* is an emerging concept to use an observer to replicate selected process state and behavior. The possibility to add value as a result of using these services must be subject to further research.

System components interoperability means the necessity of the information exchange between them. The main challenge of interoperability implementation is that information is abstract – it is knowledge describing the process in concern state and behavior, e.g. temperature in a boiler, a car speed, an account balance, etc. Obviously, abstraction cannot be processed by cyber-physical machines. It is also impossible to transfer abstraction from one place to another over the network.

Fortunately, computer science offers a workaround to address that impossibility - the information must be represented as a binary stream. In consequence, we can usually use both terms as interchangeable ones while talking about ICT systems. Unfortunately, these terms must be precisely observed in the context of further discussion, because we must be aware of the fact that the same information could have many different but equivalent representations. In other words, the same information can be represented by a vast variety of different binary patterns. For example, numbers may be represented using 2's Complement and Floating-Point binary representations.

It should be nothing new for us, as it is obvious that the same information printed as a text in regional newspapers in English, German, Polish, etc. does not resemble one another, but the text meaning should always be the same. To understand a newspaper we must learn the appropriate language. To understand the binary data we must have defined a data type – a description of how to create an appropriate bits pattern (syntax) and rules needed to assign the information (semantics), i.e. make any correct bitstream meaningful. Concluding, to make two systems interoperable, a semantic-context must be established. The type plays the role of metadata, a set of data that describes other data. Metadata term is frequently used if the semantic-context is defined using a native language to select built-in types engaging a general-purpose graphical user interface.

Using the data type definitions to describe information interchange between communicating parties allows:

- development against a type definition of the user interface
- implementation of the functionality of the bitstreams conversion in advance

Having defined types in advance, a gateway may provide dedicated conversions functionality, i.e. replacing bitstreams used by the CPS with equivalent ones for the cloud-based services. The Azure offers a vast variety of built-in types ready to be used in common cases, but not necessarily there are equivalent counterparts in use by the CPS. Additionally, the data conversion must address the following issues:

- usually to covert data from source to destination representation, the middleware software native types must be used
- if the out of the box set of types is not capable of fulfilling more demanding needs, custom data types must be defined

Although the data conversion is a run-time gateway task the implementation of the conversion algorithms must be recognized as an engineering task, and therefore this topic is not considered for further discussion.

In **IoT Central** a CPS is represented as a set of devices. The characteristics and behavior of each device kind are described by the device template. This Device Template (DT) contains also metadata describing the data exchanged over the wire with the CPS called Device Capability Model (DCM). Additionally, the DT contains properties, customization, and views definitions used by the service locally. As an option, DCM expressed as a JSON-LD can be imported into a Device Template. **IoT Central** also allows to create and edit a DCM using the dedicated web UI. A JSON file containing DCM can be derived from an information model used as a foundation to establish the semantic-context applied to achieve interoperability of the devices interconnected as the CPS. DCM development against any external information model is a design-time task and should be supported by dedicated development tools. In any case, the data interchanged between the cloud and the gateway must be compliant with the DCM, hence, the middleware software must be aware of conversions that must be applied to achieve this interoperability.

From the cloud side, it is proposed to engage the *IoT Hub* service to handle the network traffic targeting the cyber-physical system. This service offers profiles of the AMQP, MQTT, and HTTPS protocol stacks. In any case, process data is transparently transferred back-and-forth to the upper layer **IoT Central** service. Hence, the DCM associated with the **IoT Central** solution determines the payload formatting.

The Azure supports HTTP, AMQP, and MQTT protocol stacks, which are all standard ones. Consequently, it is possible to apply any available implementation compliant with an appropriate specification to achieve connectivity. In this case, all parameters required to establish semantic and security contexts are up to the gateway responsibility. Alternatively, the API offered by the dedicated frameworks (libraries) may be used. Using a framework the configuration process may be reduced significantly, and the communication protocol selection has only an indirect impact on the interoperability features. The Azure interconnection may be obtained using the following *Microsoft.Azure* packages:

- *Devices* - Service SDK for Azure IoT Devices
- *Devices.Client* - Device SDK for Azure *IoT Hub* including dependencies implementing MQTT and AMQP communication
- *Devices.Shared* - Common code for Azure IoT Device and Service SDK
- *Devices.Provisioning.Client* - Provisioning Device Client SDK for Azure IoT Devices
- *Devices.Provisioning.Transport.** - Provisioning Device Client Transports (HTTPS, AMQP, MQTT) for Azure IoT Devices

2.3 OOI Main Technology Features

To promote interoperability and address the demands of the M2M communication in the context of a multi-vendor environment the prototyping should use a framework that must be compliant with the OPC UA Part 14 PubSub (Sect. 2.1) and support the *Reactive Interoperability* (Sect. 1) concept. A framework

compliant with these requirements has been implemented as an open-source library named *UAOOI.Networking* (*Networking* for short) under an umbrella of the project Object-Oriented Internet (OOI) [16]. The library is designed to be a foundation for developing application programs that are taking part in a message-centric communication pattern and interconnected using the reactive networking concept. The diagram in Fig. 2 shows the relationship between the library (*SDK*) and external parts composing any reactive networking application (*Reactive Application*). The *Reactive Application* is an aggregation of parts implementing the *Producer* and *Consumer* roles. By design, they support access to real-time process data, hence they are recognized as an extension of *DataRepository* class. To implement the *DataRepository* dedicated implementation of the *IBinding* interface should be provided to create a bridge between CPS and an external row data represented by the *LocalResources* class. A more in-depth description of the *OOI Reactive Application* library enabling data exchange over a network using the reactive networking pattern is in [13].

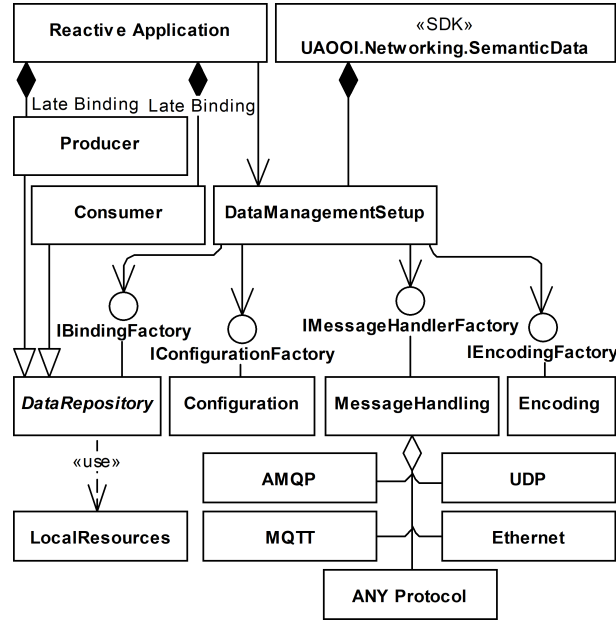


Fig. 2. Reactive interoperability implementation

From the above discussion, we can learn that the main design decisions must concern standardization and flexibility. Standardization needs the selection of an international interoperability specification to make the library ready to be adopted by the multi-vendor environment. Flexibility requires an architecture that promotes the polymorphic independent implementation of essential functions.

Piece by piece integration of a cyber-physical system using multi-vendor products requires that M2M communication employs international standards as the interoperability foundation. OPC Unified Architecture Part 14 PubSub has been selected in this respect (Sect. 2.1). By design, this standard should support the required publisher-subscriber communication pattern. It must be stressed, that by design, the specification provides only an abstract interoperability definition. Abstract means that the standard must not limit the implementation strategy.

Many parts in the domain model presented in Fig. 2 requires a polymorphic approach to implementation. To promote the polymorphic ready solution the following concepts have been adopted:

- **separation of concerns** - to allow an independent development of the parts [7]
- **dependency injection** - to allow late binding of separately implemented parts [5]

To promote the polymorphic approach, the library has a factory class called *DataManagementSetup* that is a placeholder to gather all injection points used to compose external parts. To be injected the parts must be compliant with an appropriate contract expressed as the following interfaces:

- *IBindingFactory* - bidirectional data exchange with the underlying process
- *IConfigurationDataFactory* - the configuration data access
- *IMessageHandlerFactory* - **Message** entities pushing to/pulling from the wire
- *IEncodingFactory* - searching a dictionary containing value converters

It is expected that the functionality implementation expressed by these interfaces is provided as independent external composable parts. The composition is accomplished at run time, and the effective application functionality depends essentially on reusable loosely coupled parts composed applying the dependency injection software engineering and adaptive programming concept [12].

The *DataRepository* represents data holding assets in the *Reactive Application* implementing the *IBindingFactory* interface. It captures functionality responsible for accessing the external process data from *LocalResources*. The *LocalResources* represents an external part that has a very broad usage purpose. For example, it may be any kind of the process data source/destination, i.e. raw data (e.g. PLC internal registers), OPC UA Address Space Management, cloud, file, database, graphical user interface, to name only a few.

Depending on the expected network role the library supports the implementation of:

- *Consumer* - entities processing data from incoming messages,
- *Producer* - entities gathering process data and populating outgoing messages.

The *Consumer* and *Producer* classes are derived from the *DataRepository* (Fig. 2). The *Consumer* uses the *IBindingFactory* to gather the data recovered from the *Message* instances pulled from a network. The received data may be processed or driven to any data destination, e.g. cloud-based front-end. The *Producer* mirrors the *Consumer* functionality and, after reading data from an associated source, populates the *Message* using the gathered data.

By design, the *DataRepository* and associated entities, i.e. *Local Resources*, *Consumer*, *Producer* are embedded in external parts, and consequently, the application scope may cover practically any concern that can be separated from the core *Reactive Application* implementation.

The application startup phase concerns all the parts composed to make a running instance of the application program using the dependency injection approach to allow separate development and late binding. This approach makes any modification straightforward at any development and maintenance stage. From the end-user point of view, the composition process of the running program using independently developed parts is invisible. Because in any case a single program instance is created in a typical approach, we shall expect a commonly shared configuration mechanism providing mutually exclusive parameters to separately developed parts. This scenario has been addressed by the proposed implementation thanks to making the configuration expandable.

From the discussion covered by the Sect. 2.1 we can infer that the interoperability of communicating parties requires establishing a semantic-context and security-context between them. If the received data is driven to any external data destination (e.g. cloud-based front-end) using out-of-band protocol semantic-context and security-context must be defined for this interoperability relationship. There are a vast variety of possibilities for how to define these contexts. Anyway, the most common starting point is the configuration. The proposed approach to instantiate the application as a set of composable parts is very useful in this respect because it makes it possible to expand the core configuration functionality and adapt it to be helpful also in this case. Therefore, the semantic-context may be created based on the configuration using the implementation of the *IConfigurationDataFactory* interface.

Concluding, decoupling of this functionality implementation using an abstract contract and late binding mechanism allows:

- implementation of practically any configuration management scenario,
- modification of this functionality later after releasing the library or deploying the application program in the production environment.

3 Azure - OOI Implementation

A generic domain model presenting interconnection architecture between the *Azure Cloud-based Front-end* and *Cyber-physical node* attached to the *Cyber-physical network* is presented in Fig. 1. It is proposed to implement the *Cyber-physical node* by adopting the *Reactive Application* archetype compliant with

the reactive interoperability concept (Fig. 2) described in the Sect. 2.3. Merging selected entities of this archetype into the proposed domain model (Fig. 1) lead to a model expressed as the diagram presented in Fig. 3. In the proposed approach the *Embedded Gateway* is derived from the *Consumer* role implemented as as a composable part aggregated by the *Reactive Application*.

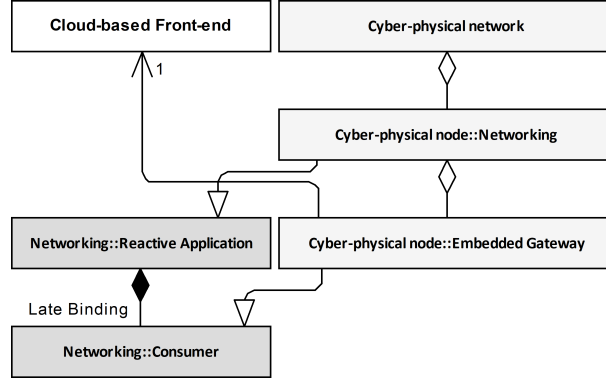


Fig. 3. Architecture Domain Model

In the final deployment architecture (Fig. 4) the *Consumer* role has been realized by the *PartDataManagementSetup* that is derived from *DataManagementSetup* provided by the library. *Networking* (SDK) was removed from this diagram for the sake of simplicity. Instantiating *PartDataManagementSetup* is the first step for bootstrapping process of the *Consumer* role functionality. This class provides an entry point to initialize all properties, which are injection points of all parts composing this role. It extends the functionality of the *DataManagementSetup* based on the following associated classes:

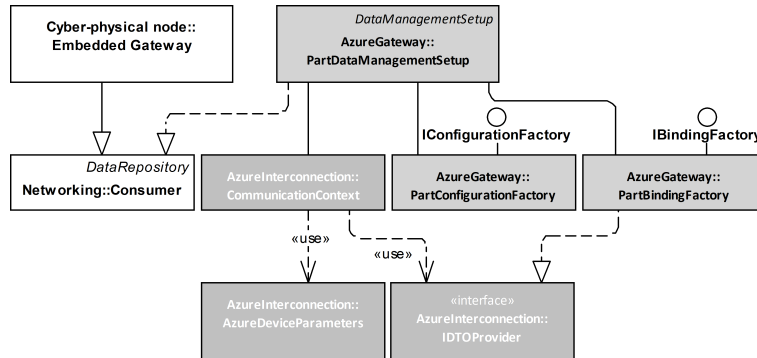


Fig. 4. Implementation Architecture

- *PartConfigurationFactory* - to get access to the application configuration
- *PartBindingFactory* - to gather the data recovered from the *Message* instances pulled from the wire
- *CommunicationContext* - to handle communication with the Azure services

In this case, the implementation of the *IConfigurationFactory* interface is based entirely on the *Networking* library. The implementation provided by the library is generic, but it is flexible enough to meet all the requirements, which warrants no need for further modifications for adoption. It is ready to provide all necessary configuration parameters to establish semantic and security contexts related to Azure interoperability.

The *PartBindingFactory* implements the *IBindingFactory* to gather the data recovered from the *Message* instances pulled from a network. The received data is driven to *CommunicationContext* for further encoding and finally pushing it to the Azure services using the configured out-of-band protocol.

IEncodingFactory and *IMessageHandlerFactory* interfaces (Fig. 2) have been implemented in independent libraries and are not considered for any modifications to deploy *Embedded Gateway* functionality. The current implementation of the mentioned interfaces is localized at run time as services using an instance of the *IServiceLocator*³. In this example, it is assumed that *IServiceLocator* is implemented to resolve references to any external services. Both interfaces are used to manage the interconnection of the node with the *Cyber-physical Network*.

The Azure interconnection is realized using the *CommunicationContext*. It implements a message encoder and establishes the out-of-band communication stack.

The recovered data from the *Message* is obtained from the *PartBindingFactory* using the *IDTOProvider* that defines a contract used to pull the Data Transfer Object (DTO) created from a selected subscription by the *PartBindingFactory*. The transfer process requires data conversion from source to destination encoding, i.e. replacing bitstreams used by the CPS with equivalent ones for the cloud-based services. The Azure offers a vast variety of built-in types ready to be used in common cases, but not necessarily there are equivalent counterparts in use by the CPS. The Azure uses JSON based Data Transfer Object encoding and schema defined based on the solution metadata. The PubSub uses JSON and binary Data Transfer Object encodings. In any case, the data recovered from the *Message* pulled from a subscription is stored locally using the object model based on standard .NET types. *PartBindingFactory* maps selected object graph onto the JSON message required by the cloud services.

The encoded JSON messages must be transferred over the network to Azure using the selected protocol stack. The Azure supports standard protocols (Sec. 2.2). Consequently, it is possible to apply any available ones compliant with an appropriate specification to achieve connectivity. In the proposed gateway implementation, the Azure interconnection has been obtained using the packages described in Sect. 2.2.

³ <https://www.nuget.org/packages/CommonServiceLocator>

Azure and PubSub use different security mechanisms so in the proposed solution establishing security-context is realized independently. The *CommunicationContext* is responsible to establish this context as an embedded negotiation phase tightly coupled with establishing interconnection.

4 Conclusion

Nowadays, the macro optimization of the industrial processes requires an integration of a vast variety of distributed applications provided by Information and Communication Technology. It requires further research on the integration of **machine - centric** Cyber-Physical Systems (CPS) with **human - centric** front-end in the context of new emerging disciplines, i.e. Industry 4.0 and the Industrial Internet of Things (Sect. 1).

CPS is composed using the multi-vendor components (data holding assets) interconnected atop of the Machine To Machine (M2M) communication. In many applications, the dynamic nature of the CPS must be considered. Dynamic nature means that interconnected assets may be added/removed from the network at any time. By design, typically CPS must fulfill the real-time and mobility of the assets requirements.

Highly-distributed solutions used to control/monitor a set of geographically spread islands of automation (e.g. virtual power plants producing renewable energy) additionally must leverage public communication infrastructure, namely the Internet. If islands of automation must be controlled over the Internet, the cloud computing concept may be recognized as a reasonable answer. Following this concept, the cloud-based supervisory control functionality is applied as a set of services employing abstraction and virtualization - two main pillars of the cloud computing paradigm. In the cloud computing concept, virtualization is recognized as the possibility to share the services by many users, and abstraction hides implementation details.

The main goal of this research is working out a new generic architecture and deployment scenario applicable for the integration of the **machine-centric** CPS and emerging cloud computing as a **human-centric** front-end.

If we must bother with the network traffic propagation asymmetry or mobility of the asset network attachment-points the reactive relationship [13] could alleviate the challenges posed by the interactive approach. The real-time multi-vendor environment makes communication standardization especially important. To support this environment, OPC Unified Architecture [9] interoperability standard has been selected. As it was pointed out in the Sect. 2.1 using OPC UA PubSub [2] the aggregation of nodes by the network is loosely coupled, i.e. nodes can be added and removed from the network dynamically, and nodes may represent mobile data holding assets.

From the analysis covered by the Sect. 2.1 it is concluded that the *Embedded Gateway* archetype best suits all requirements described above. It relaxes most of the issues related to **direct interconnection** and solutions inferred from the middleware concept, i.e. **edge device** - remote cloud agent and **field**

level gateway - a dedicated custom agent. Additionally, it could be used to connect to many independent clouds at the same time. The generic domain model for the proposed interconnection archetype is presented in the Fig. 1. We have highlighted that to promote the separation of concern design principle, the gateway functionality should be implemented as a self-contained dedicated software part embedded in the core *Networking* service of the *Cyber-physical node*. The main functionality of this component is to transfer process data between *Cyber-physical network* using *Networking* services of an existing *Cyber-physical node* and *Cloud-based front-end* using officially supported by the cloud vendor interconnection services.

To promote interoperability and address the demands of the M2M communication in the context of a multi-vendor environment the prototyping should use a framework that must be compliant with the OPC UA Part 14 PubSub (Sect. 2.1) and support the *Reactive Interoperability* (Sect. 1) concept. We proposed to use an open-source library named *UAOOI.Networking* (*Networking* for short) (Fig. 2) for this purpose. It is worth stressing that based on this approach only dedicated functionality related to the communication with the cloud must be implemented.

We derived the final model presented in Fig. 4 by merging selected entities from the *Networking* library (Fig. 2) into the generic interconnection domain model (Fig. 1). In the proposed approach the *Embedded Gateway* is derived from the *Consumer* role implemented as a composable part aggregated by the *Reactive Application*.

The proposals are backed by proof of concept reference implementations. Prototyping addresses Microsoft Azure cloud as an example. The outcome has been just published on GitHub as the open-source (MIT licensed) repository. The proposed solutions have been harmonized with the more general concept called the Object-Oriented Internet.

The described results prove that the *Embedded Gateway* archetype implementation is possible based on the existing standalone framework supporting reactive interoperability atop of the M2M communication compliant with the OPC UA PubSub standard. It is worth stressing that there is no dependency on the Client/Server session-oriented relationship. This relationship is in contrast to the architecture described in the OPC UA Part 1 [1] specification where the publisher role is tightly coupled with the **Address Space** embedded component of the OPC UA Server. The real challenge of the future work is to prove that the proposed solution is flexible enough to be used as an archetype to embed the *Embedded Gateway* component into the OPC UA Client or Server to get connected with the cloud in case of the interactive relationship.

In the proposed approach the cloud interoperability is obtained by implementing a dedicated part employing out-of-band communication only without dependency on the OPC UA functionality at all. It is worth stressing that the gateway functionality is implemented as a part - composable to the whole without programming skills. Because the part is composed at the runtime it makes

it possible to modify its functionality later after releasing the library or even deploying the application program in the production environment.

Concluding, the paper describes a proof of the concept that applying the proposed architecture and deployment scenario it is possible to integrate selected cloud services (e.g. **Azure IoT Central**) with the *Cyber-physical network* interconnected as one whole atop of the OPC UA PubSub. It is in contrast to interconnecting cloud-based front-end services with the selected *Address Space* exposed by a selected OPC UA server limiting the PubSub role to data exporter transferring the data out of the OPC UA ecosystem.

References

1. Opc unified architecture specification part 1 - overview and concepts (2017), <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/>
2. Opc unified architecture specification part 14 - pubsub (2018), <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-14-pubsub/>
3. Bansal, N.: Microsoft Azure IoT Platform, pp. 33–48 (09 2020)
4. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. *ACM Computing Surveys* **35**(2), 114–131 (2003). <https://doi.org/10.1145/857076.857078>
5. Fowler, M.: Inversion of control containers and the dependency injection pattern (23 January 2004), <https://www.martinfowler.com/articles/injection.html>
6. González, I., Calderón, A., Figueiredo, J., Sousa, J.: A literature survey on open platform communications (opc) applied to advanced industrial environments. *Electronics* **8**, 510 (05 2019). <https://doi.org/10.3390/electronics8050510>
7. Kulkarni, V., Reddy, S.: Separation of concerns in model-driven development. *IEEE Software* **20**(5), 64–69 (2003). <https://doi.org/10.1109/MS.2003.1231154>
8. Lawton, G.: Machine-to-machine technology gears up for growth. *Computer* **37**(9), 12–15 (2004). <https://doi.org/10.1109/MC.2004.137>
9. Mahnke, W., Leitner, S.H., Damm, M.: OPC Unified Architecture. Springer, 1 edn. (May 2009)
10. Meng, Z., Wu, Z., Muvianto, C., Gray, J.: A data-oriented m2m messaging mechanism for industrial iot applications. *IEEE Internet of Things Journal* **4**(1), 236–246 (2017). <https://doi.org/10.1109/JIOT.2016.2646375>
11. Pfrommer, J., Stogl, D., Aleksandrov, K., Navarro, S.E., Hein, B., Beyerer, J.: Plug and produce by modelling skills and service-oriented orchestration of re-configurable manufacturing systems. *At-Automatisierungstechnik* **63**(10), 790–800 (2015). <https://doi.org/10.1515/auto-2014-1157>
12. Postol, M.: Csharp in practice adaptive programming. Tech. rep., Lodz University of Technology (October 5 2020). <https://doi.org/10.5281/zenodo.2578244>, <https://github.com/mpostol/TP>
13. Postół, M.: Object-oriented internet reactive interoperability. In: Krzhizhanovskaya, V.V., Závodszy, G., Lees, M.H., Dongarra, J.J., Sloat, P.M.A., Brissos, S., Teixeira, J. (eds.) *Computational Science – ICCS 2020*. pp. 409–422. Springer International Publishing, Cham (2020)

14. Postol, M.: Object-oriented internet; ua part 14: Pubsub main technology features. Tech. rep. (2020). <https://doi.org/10.5281/zenodo.4361640>, <https://commsvr.gitbook.io/ooi/reactive-communication/readme.pubsubmtf>
15. Postól, M.: Computer Game Innovations 2018, chap. Machine to Machine Semantic-Data Based Communication: Comprehensive Survey, pp. 83–101. Lodz University of Technology Press, Łódź Poland (2018)
16. Postól, M.: Object oriented internet; azure gateway implementation 1.0 (Dec 2020). <https://doi.org/10.5281/zenodo.4361640>, <https://github.com/mpostol/OPC-UA-OOI>
17. Sunyaev, A.: Middleware, pp. 125–154. Springer International Publishing, Cham (2020)
18. Verma, P.K., Verma, R., Prakash, A., Agrawal, A., Naik, K., Tripathi, R., Alsabaan, M., Khalifa, T., Abdelkader, T., Abogharaf, A.: Machine-to-machine (m2m) communications: A survey. *Journal of Network and Computer Applications* **66**, 83–105 (2016). <https://doi.org/10.1016/j.jnca.2016.02.016>