JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. xx No. xx (xxxx), pp. xx–xx

Object-Oriented Internet - Azure Interoperability

Mariusz Postół[1], Piotr Szymczak[1], Clemens Vasters[2]

[1]*Lodz University of Technology*
Institute of Information Technology
ul. Wólczańska 215, 90-924 Lodz, Poland
mailto:mariusz.postol@p.lodz.pl

[2]*Microsoft*
Faculty/Department/Office Name
Postal Address with the zip-code
clemensv@microsoft.com

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

TODO list

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. It is characteristic that, in this case, 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, it is essential that any human interaction is impractical or even impossible. This interconnection scenario requires the machine to machine communication (M2M) demanding multi-vendor devices integration.

From the M2M communication concept, a broader concept of a smart factory can be derived. In this concept, the mentioned robots are only executive assets of an integrated supervisory control system responsible for macro optimization of an industrial process composed into 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 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) addi-

tionally 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 environment 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 on time relationships. Similar approaches could be applied to adopt the well known in process control industry concepts:

- Human Machine Interface (HNI)
- 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 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 [1].

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 interconnection** - cloud-based dedicated communication services allow to attache it to the CPS making up a consistent M2M communication network using an in-bound protocol stack
- **gateway based interconnection** - typical build-in communication services allows to attache the cloud computing to the CPS using an out-of-bound protocol stack

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 imposable 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 CPS in the context of new emerging disciplines, i.e. Industry 4.0 (I4.0) and the Internet of Things (IoT). The new architecture is proposed for integration of the multi-vendor **machine-centric** CPS designed atop of M2M reactive communication and emerging cloud computing as a **human-centric** front-end. To support the multi-vendor environment OPC Unified Architecture interoperability standard has been selected. The proposals are backed by proof of concept reference implementations. Prototyping addresses Microsoft Azure Cloud as an example. The prototyping outcome has been just published on GitHub as the open-source (MIT licensed). The proposed solutions have been harmonized with the more general concept called the Object-Oriented Internet.

The main goal of this article is to provide proof that:

1. 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
2. Azure interoperability can be implemented as an external part employing out-of-band communication without dependency on the OPC UA implementation
3. 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. Cloud to Sensors 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 interactive web-based user interface is required (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

1. **machine-centric** machine to machine real-time mobile interoperability
2. **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 messages payload exchanged over the network (Data Transfer Object - TDO) is standardized so that the payload 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

In practice, the set of assets embedded in the CPS is very stable. On the another hand, the virtualization of 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 repressible to register and authenticate devices exposing endpoints in the public network to allows 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, TSN, 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:

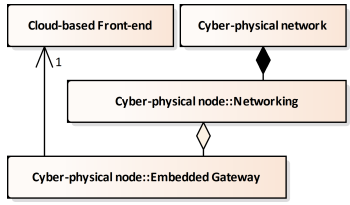

Figure 1. Strategy Domain Model

- **edge device** - remote cloud agent acting as an intermediary for nodes of the cyber-physical network
- **field level gateway** - a dedicated custom agent acting as an intermediary for nodes of the cyber-physical network
- **embedded gateway** - a software part composed into a selected node of the cyber-physical network

Edge device is a device that 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 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. The only difference compared with the **edge device** is necessity to use officially supported by the cloud vendor services to get connected. In this scenario the process data may be transferred to many clouds at the same time provided that the gateway offers this functionality.

Unlike the above described solutions, the **embedded gateway** is not derived from the middleware concept. The domain model for this archetype 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*. 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 heterogenous environments in concern. The proposed implementation is designed for 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] 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.

add ref: Mariusz Posto
14: PubSub Main Tech
tures in Object Oriente
https://github.com/mpos
UA-OOI, 2019, DOI: 1
odo.1198852

The presented proposals in the article are backed by proof of concept reference implementations. For this study, prototyping addresses Microsoft Azure cloud products. There are many reasons for selecting Azure to accomplish 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 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.

More detailed description of the selected Azure features in context of the application in concern are covered by the Sec. 2.2.

Based on the sessionless and session-oriented communication patterns examination against the IoT requirements [3] 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 spec [2]. According to proposed 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 available 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 paradigm [4] worked out in an open-source project [1]. The paper [3] 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 Object-Oriented Internet concept compliant with the OPC UA PubSub [2] 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 with regards to 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 the 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

2.2.1. Services

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 own identity) software component or module that is interfacing with selected (CPS) for data collection, analysis, and also remote control. Microsoft Azure is

[1] `https://github.com/mpostol/OPC-UA-OOI`

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 concept. The solution aggregates users, devices, services, and required additional resources scoping on a selected scenario. The solution 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.

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

Following the assumption that interconnection between the cyber-physical system 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* providing a robust messaging solution - it acts as a central message hub for bi-directional communication. 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, 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 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 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.

2.2.2. Data Interchange

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 the 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 ones as interchangeable terms 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 interchanged 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 by 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 (called telemetry) 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* allows also 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.

2.2.3. Connectivity

From the cloud side, it is proposed to employ the *IoT Hub* service to handle the network traffic targeting the cyber-physical system. This service offers profiles of the AMQP, MQTT, HTTPS protocol stacks. In any case, process data (telemetry) is transparently transferred back-and-forth to the upper layer *IoT Central* service. Hence, the payload formatting is determined by the DCM associated with the *IoT Central* solution. All the mentioned protocols are 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 connectivity and security-context is up to the external software responsibility. Alternatively, the API offered by the dedicated libraries may be used. Using this API the configuration process can be reduced significantly. Using these libraries, the selection of the communication protocol has an indirect impact on the interoperability features, including performance. The connectivity with *IoT Hub*, for example, can be obtained using:

- *Microsoft.Azure.Devices* - Service SDK for Azure IoT Devices
- *Microsoft.Azure.Devices.Client*- Device SDK for Azure *IoT Hub*
- *Microsoft.Azure.Devices.Shared* - Common code for Azure IoT Device and Service SDK
- *Microsoft.Azure.Devices.Provisioning.Client* - Provisioning Device Client SDK for Azure IoT Devices
- *Microsoft.Azure.Devices.Provisioning.Transport.Amqp* - Provisioning Device Client AMQP Transport for Azure IoT Devices
- *Microsoft.Azure.Devices.Provisioning.Transport.Http* - Provisioning Device Client Http Transport for Azure IoT Devices

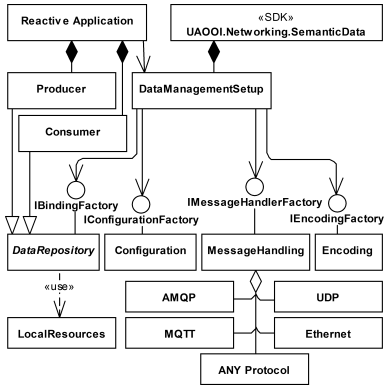

Figure 2. Reactive interoperability implementation

- *Microsoft.Azure.Devices.Provisioning.Transport.Mqtt* - Provisioning Device Client MQTT Transport 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. 3.2) and support the *Reactive Interoperability* (Sect. 1) concept. A framework compliant with this requirements has been implemented as an open-source library named *UAOOI.Networking.SemanticData* (SDK) under an umbrella of the project Object-Oriented Internet [4]. 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 *Producer* and *Consumer* parts. They must 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 [3].

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. 3.2). 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 [5],
- **dependency injection** - to allow late binding of separately implemented parts [6].

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* - pushing the *Message* entities to/pulling from *Distribution Channel*,
- *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 using the dependency injection software engineering and adaptive programming concept .

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.

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. 3.2 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 - Object-Oriented Internet interoperability Implementation

Add any information ab reusable deliverables re work.

3.1. Preface

This section describes an example implementation of an OPC UA PubSub to Azure gateway. This gateway is implemented as a composable part of the Reactive Networking Application (*RxNetworking App*). The *RxNetworking App* is an aggregation of *Producer* and *Consumer* entities derived from *DataRepository*. They must provide interconnection to real-time process data, hence they are recognized as an extension of the *DataRepository* class. *AzureGateway* part fulfills the *Consumer* role and uses out-of-band communication to push process data to the cloud.

3.2. Architecture

Domain model presenting relationship between the: Azure, PubSub Gateway, Device, Design and development tools

The *AzureGateway* functional package has been implemented based on the *Consumer* concept. This particular *Consumer* implements *IBindingFactory* interface to gather the data recovered from the *Message* instances pulled from the

Distribution Channel. The received data is driven to the Azure services using configured out-of-band' protocol. An instance of the *IBindingFactory* is responsible to create objects implementing *IBinding* that can be used by the *Consumer* to forward the data retrieved from *NetworkMessage* received over the wire to Azure services.

The proposed implementation of the Azure gateway proves that the *DataRepository* and associated entities, i.e. *Local Resources*, *Consumer*, *Producer* can be implemented as external parts, and consequently, the application scope may cover practically any concern that can be separated from the core PubSub communication engine implementation.

The article provides an introductory understanding of the steps required to implement the *Consumer* role of *OOI Reactive Application*. The *ReferenceApplication* is an example application of *Semantic-Data* reactive networking based on [OPC UA PubSub][OPC.UA.PubSub] specification. The document [OPC UA PubSub Main Technology Features][PubSubMTF] covers a description of selected fetuses relevant to this specification.

It is proof of the concept that out-of-band communication for OPC UA PubSub can be implemented based on the *DataRepository* concept.

Here are steps undertook to implement the *Consumer* role in the application:

1. *DataManagementSetup*: this class has been overridden by the *PartDataManagementSetup* class and it initializes the communication and binds data fields recovered form messages to local resources.

2. *IEncodingFactory* and *IMessageHandlerFactory*: have been implemented in independent libraries and *Consumer* doesn't depend on this implementation - current implementation of the interfaces is localized as services using an instance of the *IServiceLocator* interface.

3. *IBindingFactory*: has been implemented in the class *PartBindingFactory* that is responsible to gather the data recovered from the *Message* instances pulled from the *Distribution Channel*. The received data is driven to the Azure services using configured out-of-band protocol.

4. *IConfigurationFactory*: the class *PartConfigurationFactory* implements this interface to be used for the configuration file opening.

3.2.1. *DataManagementSetup* implementation

The *PartDataManagementSetup* constructor initializes all properties, which are injection points of all parts composing this role.

In this example, it is assumed that *ServiceLocator* is implemented to resolve references to any external services.

Finally the *DataManagementSetup.Start()* method is called to initialize the infrastructure, enable all associations and start pumping the data.

3.2.2. *IBindingFactory* implementation

Implementation of this interface is a basic step to implement *Consumer* functionality. The *DataRepository* represents data holding assets in the *RxNetworking App* and, following the proposed architecture, the *IBindingFactory* interface is implemented by this external part. It captures functionality responsible for accessing the process data represented by the *LocalResources*. The *LocalResources* represents the external part that has a very broad usage purpose. For example, it may be any kind of process data source/destination, and to name a few *Raw Data*, *OPC UA Address Space Management*, and *Azure* services in this case.

3.3. Configuration

3.3.1. *IConfigurationFactory* implementation

Implementation of this interface is straightforward and based entirely on the library UAOOI.Configuration.Networking available as the NuGet package. In a typical scenario, this implementation should not be considered for further modification. The only open question is how to provide the name of the file containing the configuration of this role.

3.4. Protocol selection

From the description covered by the Sec. and the Sec. the Azure supports the HTTP, AMQP, MQTT, but the PubSub Ethernet, UDP, AMQP, MQTT. If the Ethernet or UDP has been selected to build interconnection based on PubSub direc interoperability with the PubSub is impossible because the Azure doesn't offer these protocol as native communication services.

3.5. data mapping

For both the schema is different Data Transfer Object encoding

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 encoding. If JSON is used possibility to establish semantic context depends on the Azure metadata definition. In case PubSub uses binary encoding establishing interconnection is impossible.

3.6. Security

Azure and PubSub uses different security mechanism so establishing directly security context is impossible at all

3.7. Deployment phases

- Design
- Gateway and devices registration
- Authentication
- Device/Service association
- Device/Application association
- Establishing session
  - Device/Device Template (Device Capability Model) association - establishing a semantic-context
  - Security management - establishing security-context
- Interconnection - exchange of data
- Maintenance

4. Conclusion

The OPC UA PubSub to Azure gateway (*AzureGateway*) implementation has been just published on GitHub as the open-source (MIT licensed) as a part of the more general concept of the Object-Oriented Internet reactive networking. It is proof of the concept that

1. OPC UA PubSub can be implemented as a powerful standalone package - no C/S dependency
2. Azure interoperability can be implemented as an out-of-band communication (MQTT, AMQP, HTTP) - no PubSub dependency
3. Process data functionality can be composable at run-time - no programming required

References

[1] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A. M., *The Many Faces of Publish/Subscribe*, ACM Computing Surveys, Vol. 35, No. 2, 2003, pp. 114–131.

[2] *OPC Unified Architecture Specification Part 14 - PubSub*, 2018.

[3] Postół, M., *Object-Oriented Internet Reactive Interoperability*, In: Computational Science – ICCS 2020, edited by V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, Springer International Publishing, Cham, 2020, pp. 409–422.

[4] Postol, M., *Object-Oriented Internet*, Tech. Rep. 5.1.0, 2019.

[5] Kulkarni, V. and Reddy, S., *Separation of concerns in model-driven development*, IEEE Software, Vol. 20, No. 5, 2003, pp. 64–69.

[6] Fowler, M., *Inversion of Control Containers and the Dependency Injection pattern*, 23 January 2004.