Häme University of Applied Sciences - HAMK
University Centre – Hämeenlinna
Smart Services Research Unit

From Edge to Cloud: the issues with connectivity related to Internet of Things

Keywords: IoT, Data, Cloud, *ThingWorx*, *Azure*
Project developed under the exchange program *PROPICIE 10th ed.*, in cooperation between the Brazilian institution *Federal Institute of Santa Catarina - IFSC* and the Finnish institution *Häme University of Applied Sciences - HAMK*.

Advisors:
Vinícius Berndsen Peccin (IFSC)
Joni Kukkamäki (HAMK)
Author:
Leonardo Drews Montibeller

Hämeenlinna - FI, January 2017

Abstract

Internet of Things is a quite new field of study and it is proving its value. Still, is not clear how is best way to deploy one IoT solution. In all the steps of the path of data is possible to find many problems, with many solutions. This project explores the way the devices are connected to the cloud, with a brief study of business cases of IoT and tests the connection to two major cloud platforms, *ThingWorx* and *Azure*.

Table of Contents

1. INTRODUCTION

A good management rely in a good supervision, and therefore, in a good control of your process. That is well known in the field of automation: one can only has a efficient control of your system if it sees efficiently the system itself. When talking about personnel administration , a manager needs to know exactly what the skills of the employees are and how they are performing. If someone is not performing well, the manager needs to know why. Humans are complex creatures and often is really hard to measure their performance and even harder to determine why the performance is not satisfactory. However, the scenario is completely different when referring to the capabilities of humans measuring the physical variables of the world.

In industry, many variables are measured constantly: temperature, speed, humidity, power, etc. A lot of money, time and effort are spent in refining supervision and control systems of plants. If something is too hot, a fan starts cooling it down. If too cold, a heater starts up. Sensors are constantly getting data and actuators are doing action based on that. But how much of this data is stored and analyzed through the long term? How to make that data gathered from the sensor reach the top level management? And how to make it relevant to the management? For these questions, a recent concept has appeared, the Internet of Things (IoT).

2. OBJECTIVES

This project purpose is to understand what is IoT and its role in the industrial management level. However, focused in the way things are connected and exploiting the more appropriate ways to do so in each case scenario.

Main Objective:

Understand the concept of Internet of Things and the way devices are connected to the "cloud".

Specific Objectives:

-Study of Industrial Internet of Things (IIOT).

-Study of the scenarios which IoT can be used.

-Study of connection to some of the currently available clouds for IoT: ThingWorx and Azure's IoT HUB.

-Make a practical implementation of connecting a real-case device to one of the clouds.

3. Theoretical Approach

Before starting the project, a strong theoretical base in the subjects related to Internet of Things, Data Science and Information Technology is needed. The fundamentals are:

3.1. The Internet of Things

According to the *Industrial Internet Reference Architecture* document, organized by the *Industrial Internet Consortium*, the industrial internet of things is:

> [...]An internet of things, machines, computers and people, enabling intelligent industrial operations using advanced data analytics for transformational business outcomes. It embodies the convergence of the global industrial ecosystem, advanced computing and manufacturing, pervasive sensing and ubiquitous network connectivity. [...] The Industrial Internet effort will bring industrial control systems online to form large end-to-end systems, connecting them with people, and fully integrating them with enterprise systems, business processes and analytics solutions. These end-to-end systems are referred to as Industrial Internet Systems (IISs). Within these IISs, operational sensor data and the interactions of people with the systems may be combined with organizational or public information for advanced analytics and other advanced data processing (e.g., rule-based policies and decision systems). The result of such analytics and processing will in turn enable significant advances in optimizing decision-making, operation and collaboration among a large number of increasingly autonomous control systems. (IIRA, 2015, p. 8-9)

Here it's possible to see two importants concepts of IoT. First thing, it is related to people, so it means that the data collected must be presented to human beings in a way they understand. This is important because it refers to data visualization, a field of data science strongly related to IoT. Second one is the usage of IoT to optimize "decision-making, operation and collaboration among a large number of increasingly autonomous systems". This is related to data analytics, another very important subject of data science.

How data flows and is organized through its path is another relevant topic to be known.

3.2. The path of data.

From the image shown next is possible to see what usually is the path that data takes:

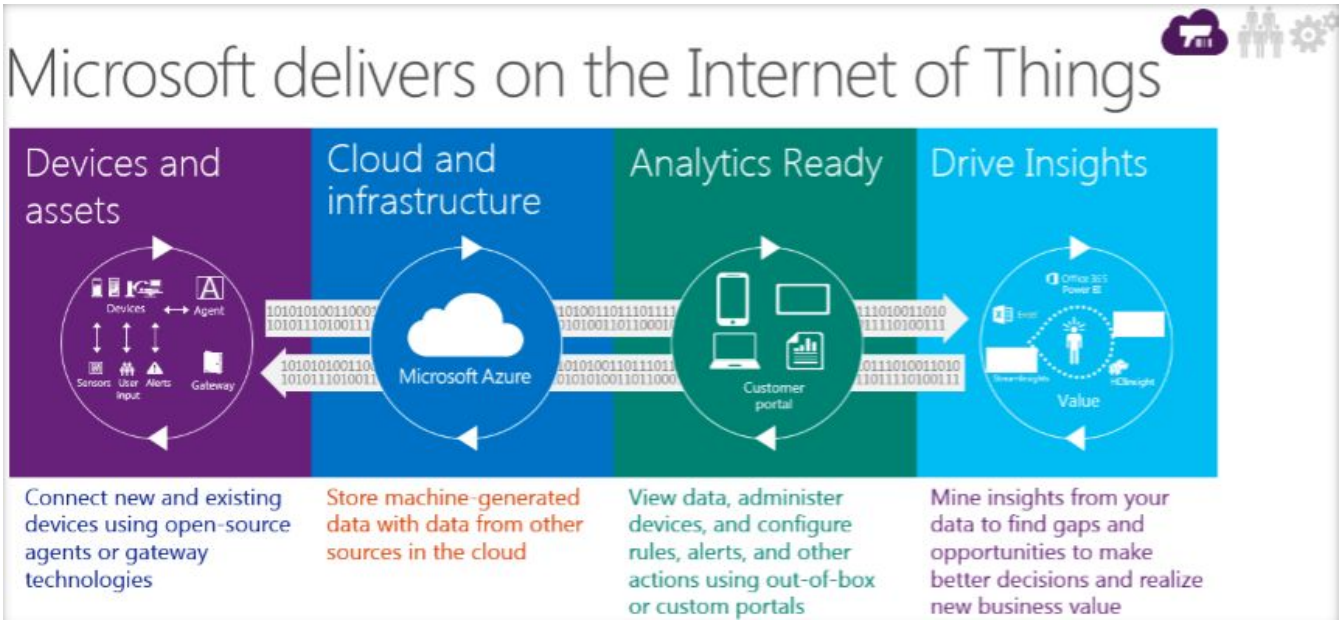

Figure 1. Microsoft delivers on the Internet of Things by Stefan Hoppe, 2015.[1]

The data start being generated as soon a device is installed. Before a sensor starts collecting the actual data, is important to keep track of this device in the production chain. Therefore questions like what is the device, where it is located, when it was deployed and how it is current status are data that must be well organized. This is important for maintain the scaling capability of one solution. According to the *National Information Standards Organization* - *Niso* (2004), all these type of data that "describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource" is called **Metadata**.

Only after that, the raw data itself can be delivered to a place to be stored using a safe connection. From the picture, is possible to use open-source agents and gateways technologies to connect, not only new devices, but also existing running devices to data storage place. This is the main focus of this project: understanding the **Data Acquisition** in IoT. Devices in this side of the data production chain are called **Edge** devices**.**

After the data is being acquired it needs to be stored, usually in the **Cloud**. The servers that handle the incoming stream of data are called Clouds. They can be a paid service offered by big companies, like *Microsoft's Azure, IBM's Bluemix* and *PTC's ThingWorx*. But one cloud could also be any server machine in any company. It is usual to think that clouds are something volatile and the data can never be lost, when in fact are just big data centers somewhere in the world.

Every cloud has it owns particularities, they are made for different purposes and each one offers different solutions. Here, were explored a bit of *ThingWorx* and *Azure IoT HUB*.

[1] From "IoT und Industrie 4.0: Unterschiede und Lösungen mit der Microsoft-Plattform", by Stefan Hoppe, Microsoft Technical Summit 2015. Available at https://channel9.msdn.com/Events/microsoft-techncial-summit/Technical-Summit-2015-The-Next-Level/IoT-und-Industrie-40-Unterschiede-und-Loesungen-mit-der-Microsoft-Plattform

The data must be, many times, filtered, fitted and treated before anything can be done. This step will lead to the **visualization** and/or the **analysis** of the data. **Data visualization** is the big amount of data translated into a human friendly form, usually using charts, graphs, diagrams, etc.

Data analytics is the refinement of the data that lead to decisions, insights or any relevant meaning from the data. In this stage is where terms like Artificial Intelligence, Data Mining, Machine Learning often are referred, and where Statistics really shine.

Worth noting in the picture that the path of data through IoT is a **two-way connection**. It means that after making the decision, another set of data is used to actuate in the original data, changing them and probably reaching the objectives.

Still, how strong is this approach? How much it worth using?

3.3 Usages of IoT

Still not really clear what are best uses of IoT. However, there are already some successful cases and the market is learning where to aim. Some areas are more suitable, like shown in the graph below:

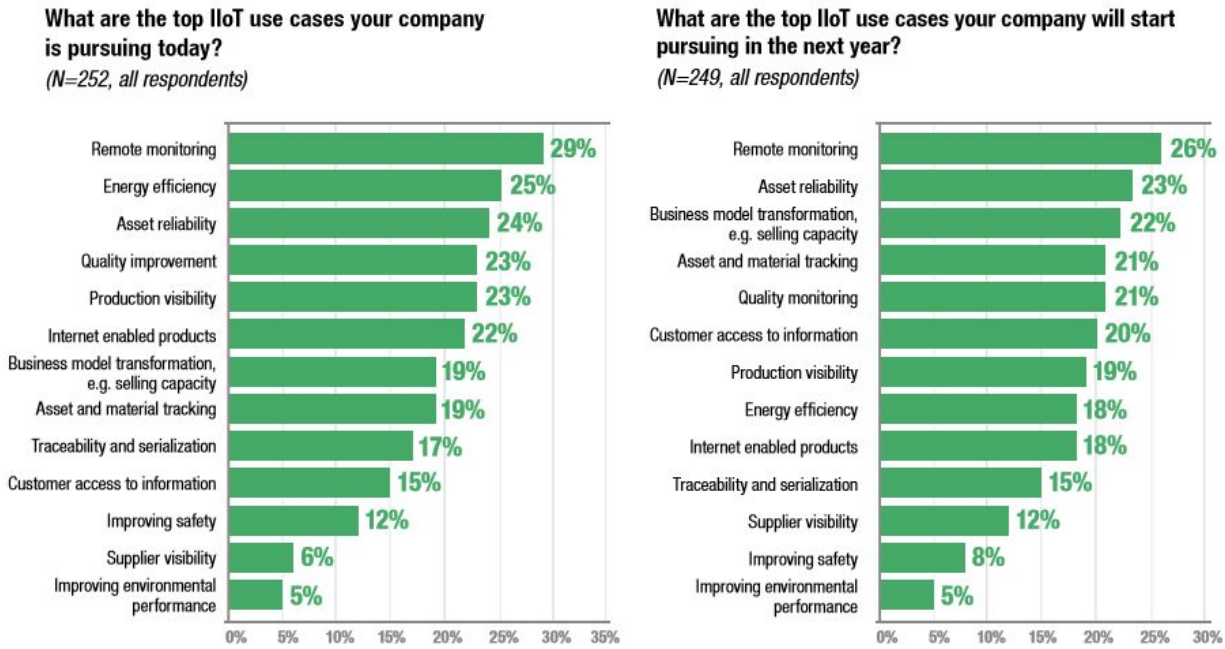

Figure 2. IIoT cases. (Source: LNS research, 2016).[2]

[2] From "IIoT AND BIG DATA ANALYTICS: How Manufacturing System Architecture Is Being Transformed", by Matthew Littlefield, LNS research, 2016. Available at http://blog.lnsresearch.com/iiot_and_big_data_analytics__transforming_manufacturing_system_architecture

Remote monitoring is at the top as an obvious purpose for IoT in industries. That is what comes in mind when we think about connecting things to the internet, but the interesting part is what is possible through remote monitoring.

ThyssenKrupp is a leading company in elevator business. An article[3] tells how the elevator company wanted to change its business model from reactive to predictive. Before, probably, the company either had to wait for a call telling that one elevator was malfunctioning, or do periodical maintenance to minimize failures. After investing in remote monitoring, *ThyssenKrupp* managed to reduce downtime and costs with maintenance, being even able to "replace components and systems before the end of their lifecycle."[4]

4. Methodology and resources

After understanding the basics concepts and why IoT is valuable, the first weeks of the project were spent getting familiar with the resources used in the research unit. The first cloud platform to be used was *ThingWorx*.

4.1. The *ThingWorx Enterprise IoT Platform*.

Using *ThingWorx* was only possible due the partnership between HAMK and the finnish communications company *Elisa*. Concerned about being in constantly innovation, supplying the newest and best experience to its customers, *Elisa* has provided the platform, free of charge, for HAMK's Smart Services Research Unit.

4.1.1. The *ThingWorx* model.

The first thing to be noticed is that *ThingWorx* is made specifically for fast development and deployment of IoT solutions. This means that the platform is overall simple, with a small necessity for coding, which means that the development time is spent more in understanding the tools and configuration. Also, such purpose shown some limitations and a learning curve that is easy for simple things but very complex for more elaborate solutions.

Second, the platform is entirely *web* based. Very handy for IT developers working in the cloud side, but very unusual for engineers working with the edge devices. With very small background knowledge in web development, my first task in the project was to understand the *ThingWorx* model and its functioning, along with some of technologies which the platform makes use of.

[3] MASSON, Collin. "From commoditization to servitization: Transforming your business to compete in the new age of field service with IoT", (2016, November 7), *Microsoft*. Available at https://blogs.microsoft.com/iot/2016/11/07/from-commodization-to-servitization-transforming-your-business-to-compete-in-the-new-age-of-field-service-with-iot/

[4] ThyssenKrupp. "MAX - Predictive Maintenance Solution". 2017. Available at https://max.thyssenkrupp-elevator.com/en/

Putting in simple terms, *ThingWorx* is a website which uses webservices to connect devices, collect, process, visualize and analyse the data. Pretty much what IoT is about. The big difference of this platform is that, in order to make it simple and fast-deployment, it uses it owns model of entities representation:

> ThingWorx Model Overview
>
> - Things may represent physical assets, people, organization elements (departments or production lines), and parts of a work process.
> - Things can contain data, can send and receive data, and can perform Services.
> - Things may fire Events.
> - Things may subscribe to Events from other Things in your model.
>
> After you have your model in place, you can assemble the data, services, and capabilities of the model into a web application via the drag and drop Mashup Builder.
>
> (PTC , 2016 *)[5]

There are entities types for physical things (devices, networks, model tags), for analytics, visualization (mashups, dashboards, gadgets, menus), data storage, collaboration, management, security, server services and many more that can be imported to the platform. The paradigm here is like Object-Oriented Programming: objects with properties and methods that can be used in different ways. The portal (main page) where the user can create their own solution is called the *composer*. A image of the *composer* interface can be seen below:

[5] From "ThingWorx 7.0 Help Center". PTC. 2016*. Available at http://support.ptc.com/cs/help/thingworx_hc/thingworx_7.0_hc/

*The documentation does not have a release date, the year was based on the release notes available at https://community.thingworx.com/docs/DOC-3131

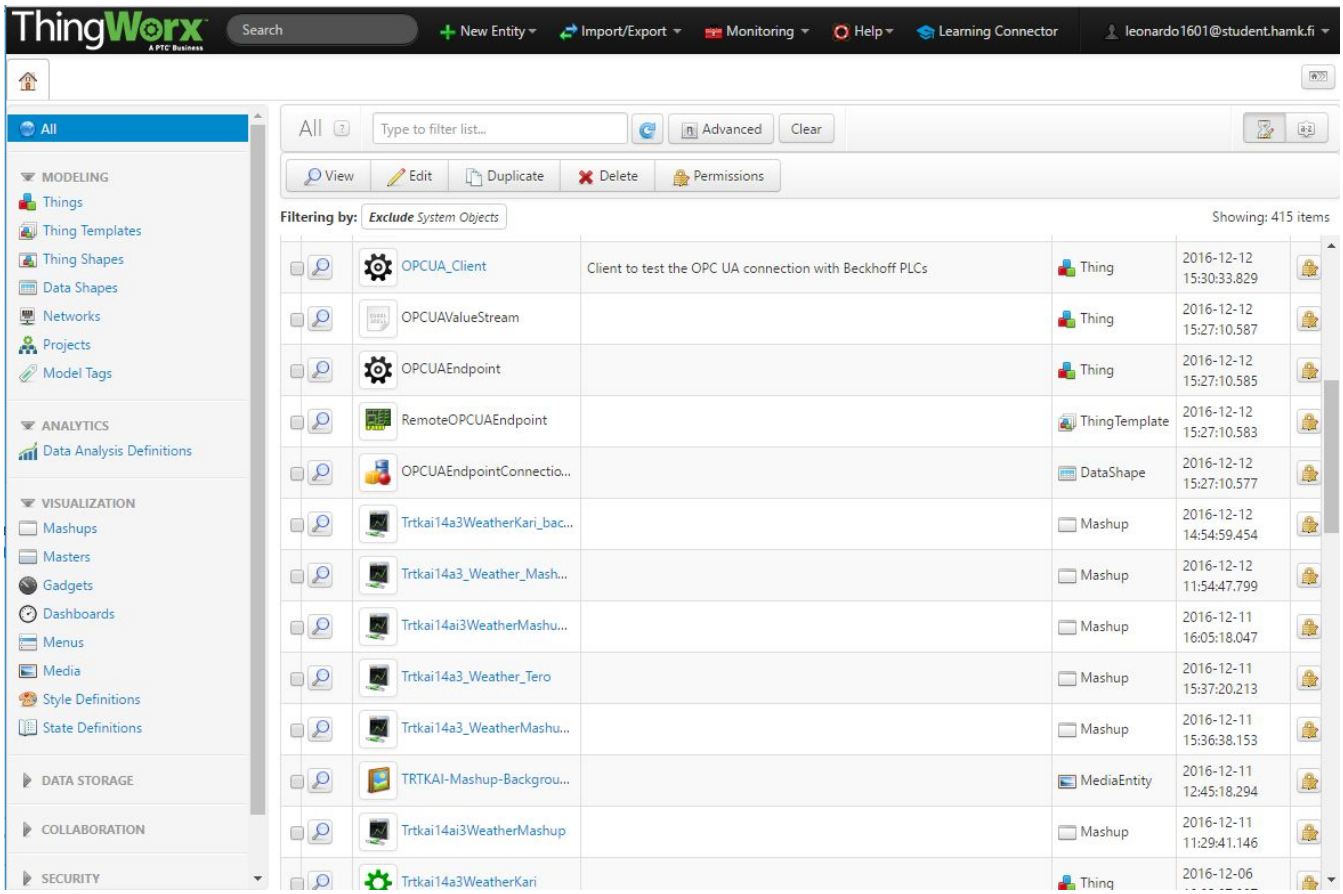

Figure 3. ThingWorx Composer user interface.

4.1.2. The *ThingWorx* REST API.

One of the tools for communication that the platform use is the Representational State Transfer (REST) web services. REST is an easy way to work with web services and can be used by any client that is capable of making an HTTP request. More information about how to use the API can be found following the link below:

- *ThingWorx REST API Cheat Sheet (2016):* https://community.thingworx.com/docs/DOC-3315

In case the reader is not familiar with terms like API or HTTP, material giving a brief explanation about these topics can be found in *Appendix I* , under the section *Appendices,* in the end of this paper.

4.1.3. The connection to *ThingWorx*.

In order to a device send and receive data to the platform, it needs to be able to use HTTP. The almost real time connection occurs because of the Websockets capacity of the cloud. Websocket is a

communication protocol that establish a socket connection between a client and server. The connection is persistent and both parts can send and receive messages at any moment.

WebSockets represent a standard for bi-directional real time communication. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. Firstly in web browsers, but ultimately between any server and any client. Connection limitations are no longer a problem since WebSockets represent a single TCP socket connection and its only relationship to HTTP is that it's handshake is interpreted by HTTP servers as an **`5h_jY\]`** request.

Is important to notice that *ThingWorx* has a tool called Edge MicroServer. It is a WebSocket-based server and has it own SDK (Software Development Kit) in many languages, including C, Java and .NET. It is possible to use them to deploy applications and exchange data with the cloud.

Being C the programming language that I'm most familiar with, the first connection practice was using a *Raspberry Pi* board computer and the *ThingWorx Edge* C SDK. The practice was based in one example found in the *ThingWorx* Community Forum, it can be found in the following link:

- *Using the C SDK to Deliver Data to ThingWorx from a Raspberry PI* (2015): https://community.thingworx.com/community/developers/blog/2015/12/30/using-the-c-sdk-to-deliver-data-to-thingworx-from-a-raspberry-pi

Before explaining how the connection is established, there are a few characteristics that are indispensable for IoT systems:

> Common system characteristics include upholding privacy expectations, reliability, scalability, usability, maintainability, portability and composability but three are key to the discussion of IISs because of their criticality in ensuring the core functions, rather than the efficiency, of these functions, of the system:
>
> Safety: the condition of the system operating without causing unacceptable risk of physical injury or damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment.
>
> Security: the condition of the system operating without allowing unintended or unauthorized access, change or destruction of the system or the data and information it encompasses.
>
> Resilience: the condition of the system being able to avoid, absorb and/or manage dynamic adversarial conditions while completing assigned mission(s), and to reconstitute operational capabilities after casualties. (*IIRA*, 2015, p. 11)

It is clear to see that security is a major concern. In order to achieve this requirement, connections to cloud usually use certificates. Since *WebSockets* through HTTP is the tool used here, to make it safe, *ThingWorx* uses the SSL (Secure Socket Layer)/TLS (Transport Layer Security) cryptographic protocols. Usually, when *WebSockets* are used over SSL/TLS they are called *WebSocket Secure* (WSS).

The main code of the example mentioned before can be found in my private *Github* repository:

- *TWlearning* (2017): https://github.com/ldmontibeller/TWlearning

With a closer look into the code, can be noted that, in order to access the domain of the server and establish the connection, the device must provide one Application Key (*appKey)*. This is known in SSL/TLS certification as a symmetric encryption, in this case both server and client uses the same key to validate the request. Thus, the encryption key must either given to the device by a secure wire or copied manually.

In the following images, the code, the prompt messages of the message transmission and the user interface made in the *ThingWorx* platform can be seen:

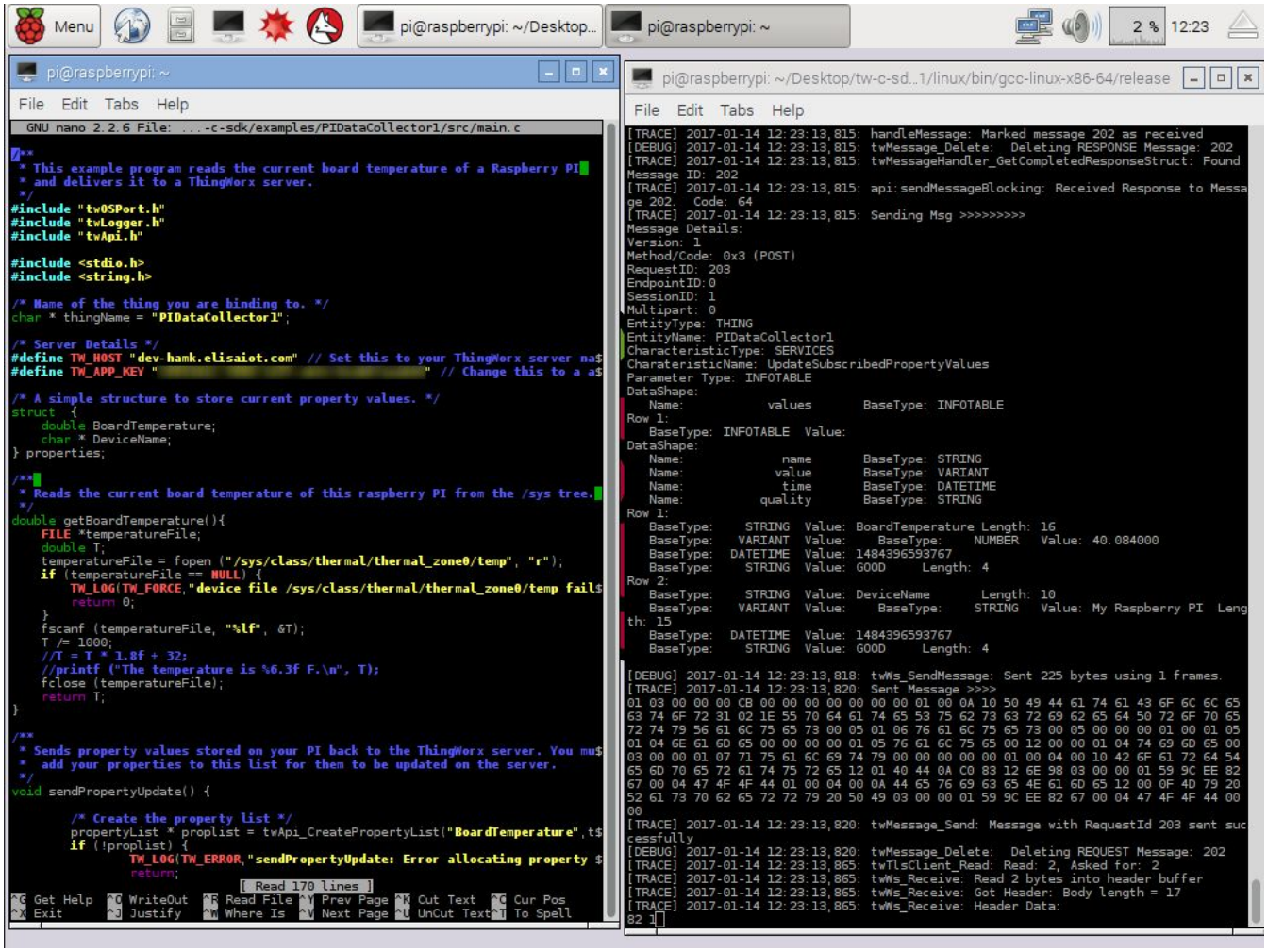

Figure 4. Code and prompt response of the given example, running in a *Raspberry Pi 3*.

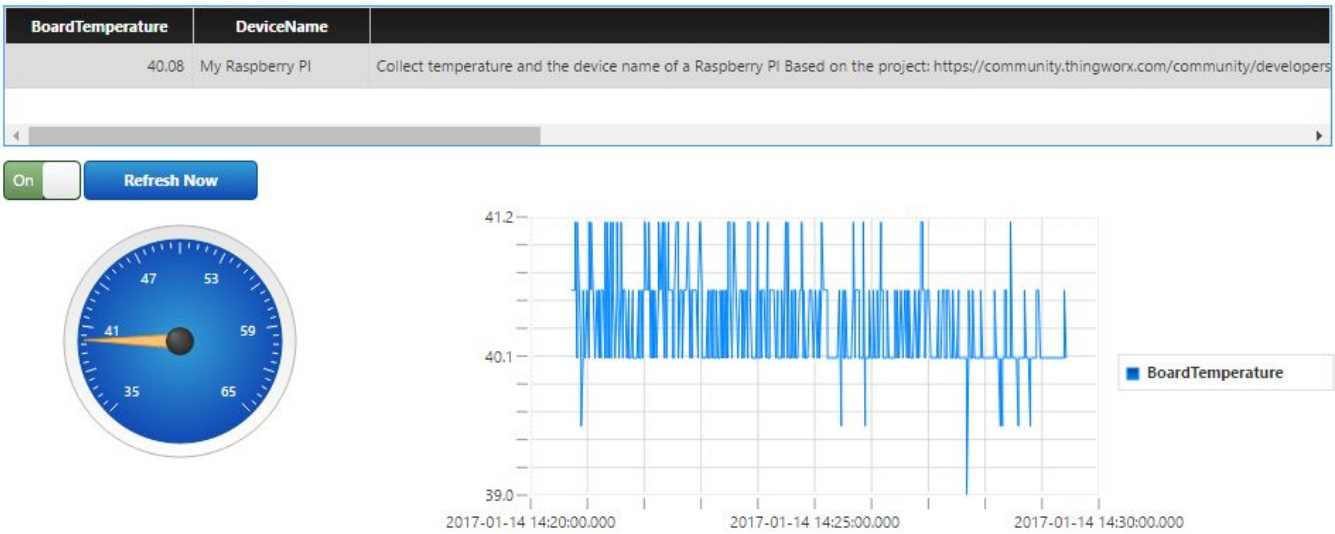

Figure 5. User interface for the given example, running in the platform.

However, what happens if the device itself does not have the ability of making HTTP requests? There are many devices that can't perform such connection because they have not the interface and processing power to do so (sensors, microcontrollers, actuators), or the device operation is limited to a different protocol (*Modbus, OPC UA)*. *ThingWorx* offers the possibility of buying a third-party solution, the *Kepware KEPServerEX.*

KEPServerEX is a full *ThingWorx* compatible software that can be installed in most of *Windows* versions, it works as a big gateway, translating the protocols to be used in the cloud. It has drivers for many devices used in industry (PLCs, Softstarters, IPCs), supporting the different communications protocols which the devices use. This come very handy in the automation field, since the reality is a that are lot of established networks, working reliably for many years now.

The investigation of the real cases in industry and business cases using IoT leads to the next section of the project.

4.2. *IoT devices usage scenarios.*

After using a *Raspberry Pi*, the focus of the project has turned to look into real cases of IoT usage in business. I've attended to meetings of another project being developed in HAMK with a partnership with *Tekme*, a building management company in Finland.

Tekme uses PLCs to get data from its customer buildings, using the data for a better management service. In those meeting I've realized some issues involving the connectivity of the PLCs and how the solutions are divided:

Figure 6. IoT Scenarios.

In the figure, old devices means the devices that cannot connect directly to the cloud, while new devices means the ones which can. Small applications are the ones who usually use only one device to acquire and manage data (e.g. 1 PLC), and the big applications are those which requires another device to handle multiples devices sending data (e.g. a server managing multiple PLCs in one industry).

Every scenario has its own peculiarities. Devices that can not connect directly to the cloud may need to gateway or another solution to get them online, in small scenarios may be more viable buy new ready-to-cloud devices. However, automated companies usually have legacy systems that have been proven safe and reliable, have required a lot of investment, making the replacement with new devices impracticable. This means that any solution which makes the connection of such systems simplier have an inherent value to the market.

And *Microsoft* has seen that.

4.3. *Microsoft's Azure* cloud platform.

Azure is the given name for a variety of services that are together hosted in the *Microsoft*'s cloud. *IoT Hub* is the cloud service that manage and establish two-way communication with the devices.If only one way is needed, just receiving data to the cloud, there is another service called *Event Hub*. To reach the hub sometimes is needed a field gateway to translate and filter the data before going to cloud.

4.3.1. *Azure IoT Hub*.

The *IoT Hub* is related to the cloud end of the architecture. Is the place that all the data cross and it is capable of:

- Two-way communication (edge < > cloud), including file transfer and request-reply methods;
- Message routing to other *Azure* services;
- Queryable storage for metadata and synchronized state information;
- Security: per-device credentials and access control;
- Monitoring: identity management of the devices and events;
- Libraries: includes device libraries for most popular languages and platforms;

This means that the cloud here already provide some pre-made tools and resources, so the customer does not need to spend time developing those. The following image will make more clear where is the place of the *IoT Hub* in the data path:

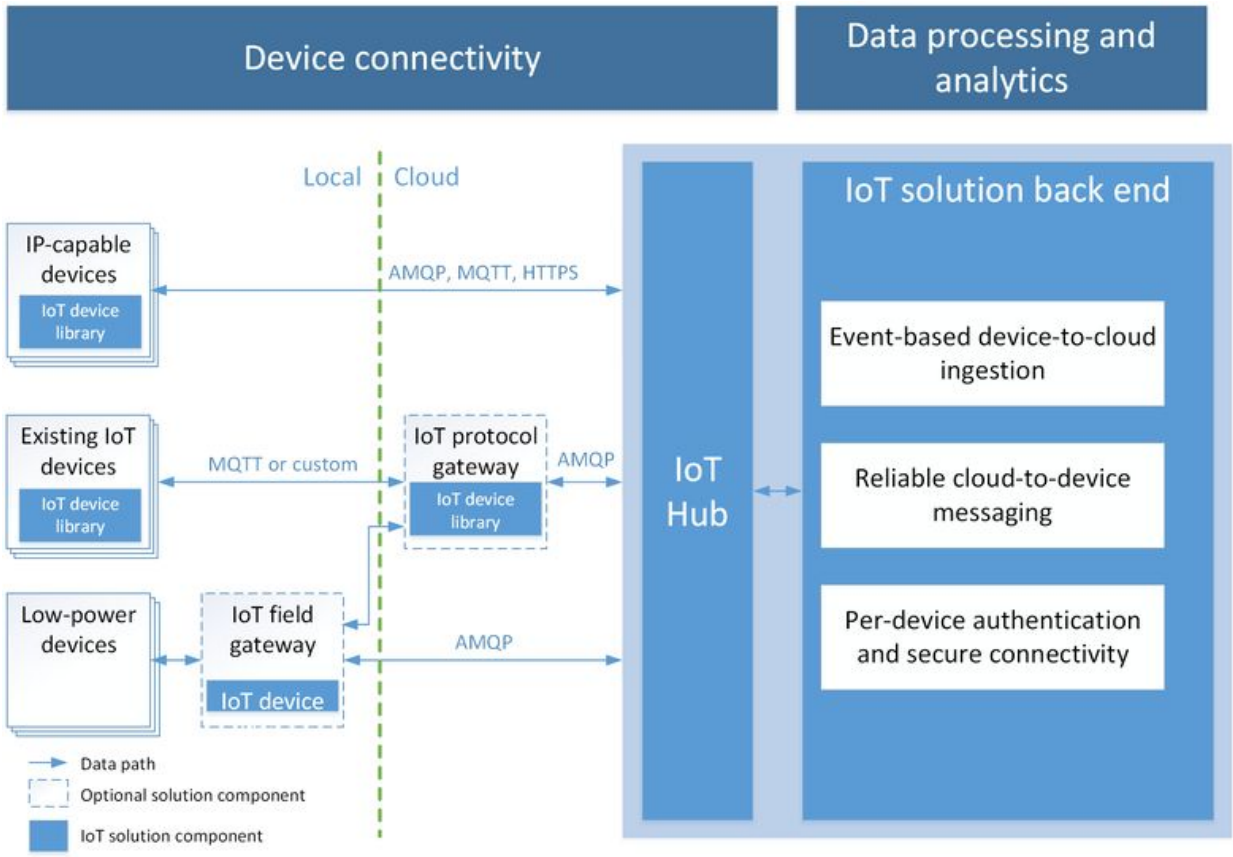

Figure 7. Hub architecture. (Source: Microsoft Azure, 2016[6])

Microsoft is trying to solve connection with the a grand variety of protocols using a cloud or a field gateway. There are some advantages of using a field gateway, this is why the focus of the company is now in the *Azure IoT Gateway SDK*.

NOTE: In order to create your devices in the IoT Hub of Azure, you can use the REST API methods or use the DeviceExplorer application available at: https://github.com/Azure/azure-iot-sdks/blob/master/doc/manage_iot_hub.md#device-explorer

4.3.2. *Azure IoT Gateway SDK.*

[6] From "What is Azure IoT Hub", 2016. Available at https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-what-is-iot-hub

As can be seen in the gateway description in its *Github* documentation (https://github.com/azure/azure-iot-gateway-sdk):

> IoT scenarios vary drastically between industries and even between customers within the same industry. The Azure IoT Gateway SDK lets you build IoT solutions tailored to your exact scenario. Connect new or existing devices, regardless of protocol. Process the data in an on-premises gateway using the language of your choice (Java, C#, Node.js, or C), before sending it to the cloud. Deploy the solution on hardware that meets your performance needs and runs the operating system of your choice. (*MICROSOFT,* 2016)

From the excerpt is possible to notice how the *Azure IoT Gateway* fits is the scenarios gap where is needed to connect old and new devices. The architecture also englobes data filtering in the edge so only real relevant information is uploaded.

The SDK provides the tools to develop a custom gateway, which makes possible to use it in many different devices. It works based in modules that send messages to exchange data with other modules. A module receives a message, performs some action on it, optionally transforms it into a new message, and then publishes it for other modules to process, until the point the message is sent to the cloud/edge. To accomplish this, the SDK has a broker that connect the multiple modules, they publish messages to the broker (bus, pubsub, or any other messaging pattern), so the broker route the message to the correct module .The message consists of a set of key/value properties and content passed as a block of memory. (STREET, 2016)

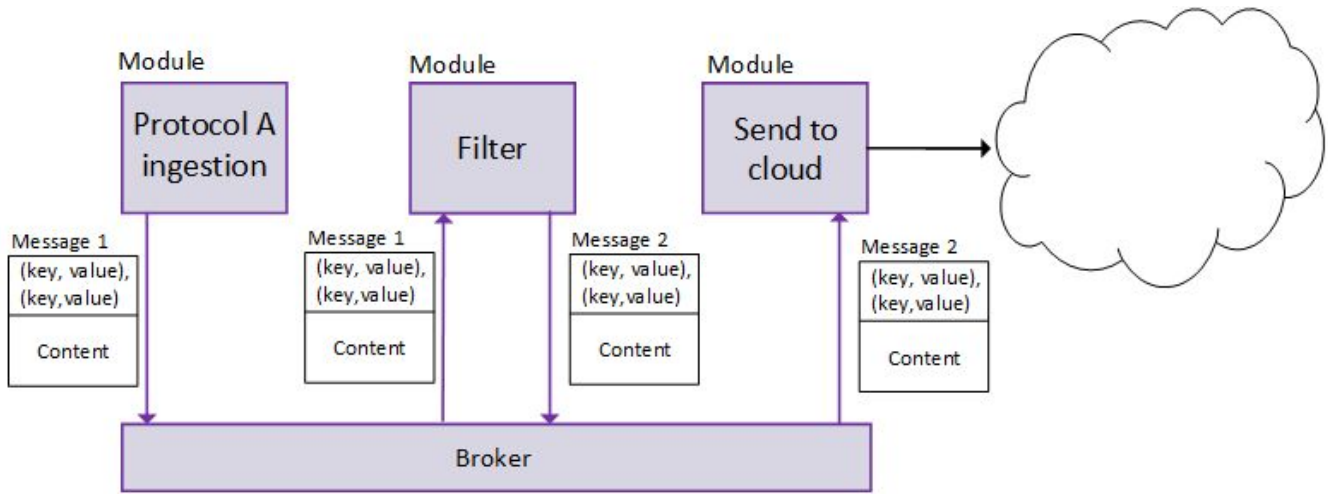

Figure 8. Message routing. (Source: Microsoft Azure, 2016[7])

. Currently, there are third-party modules being developed, like the one for *Modbus* communication protocol and *OPC UA*. Also, many real gateways to cloud that uses the SDK are already being commercialized, they can be found at:

[7] From "Get started with Azure IoT Gateway SDK", 2016. Available at https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-windows-gateway-sdk-get-started

- *Azure Certified for IoT Device Catalog* (2017): https://catalog.azureiotsuite.com/

Below is a picture of the *IoT Hub* showing telemetry data acquired from simulated devices running on a computer:

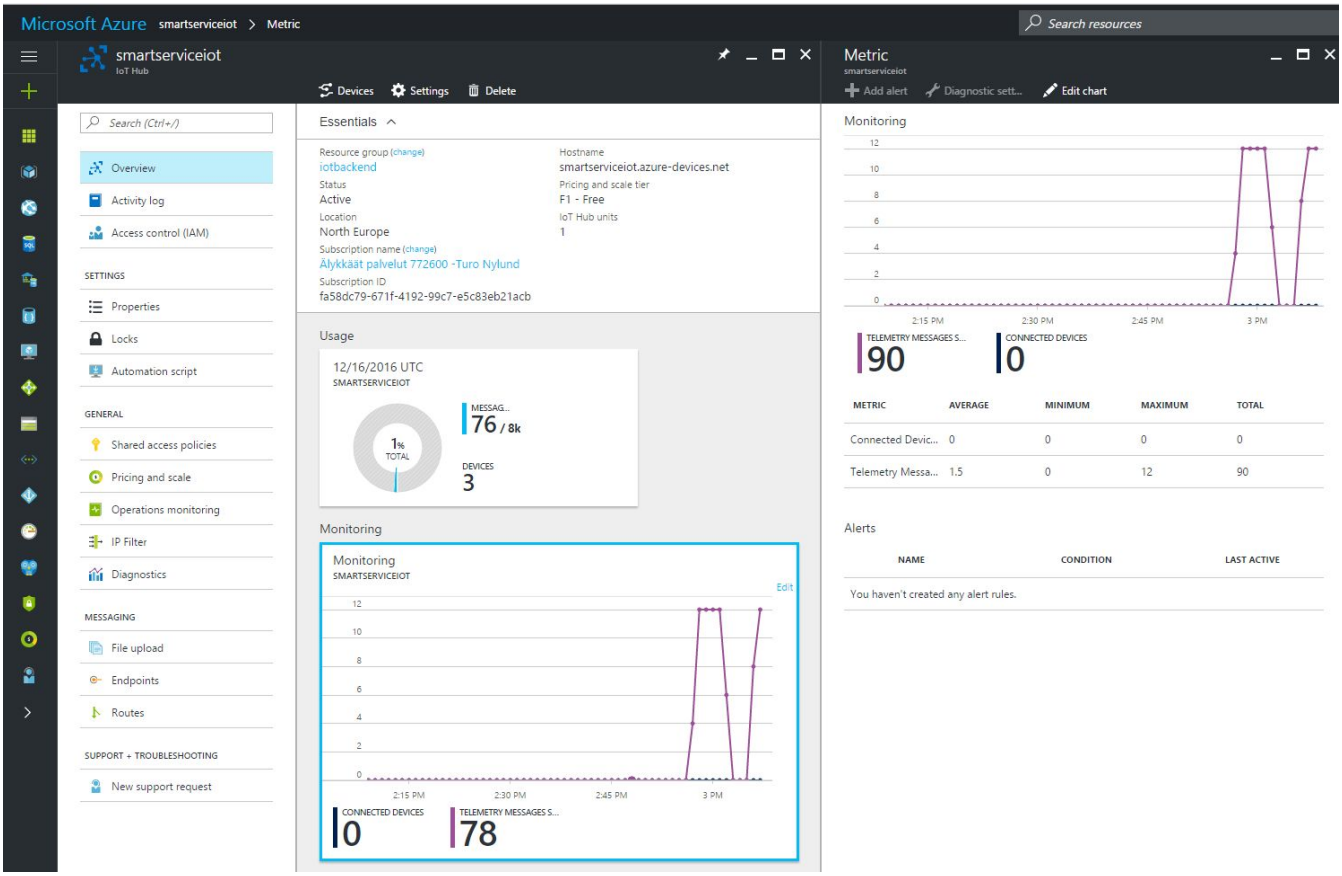

Figure 9. *Azure IoT Hub* telemetry data.

4.4. Legacy devices to the cloud.

After exploring the cloud tools, the next step of the project was the connection of a device used in the automation field. Since the Automation Engineer course of HAMK works with *Beckhoff* products, one legacy PLC was chosen, the CX9020. It is not able of natively performing *HTTP* requests, therefore it has no direct connection with *ThingWorx* or *Azure,* even though it has an *Ethernet* port, the communications protocols and the data structure are different from the ones supported by the platforms.

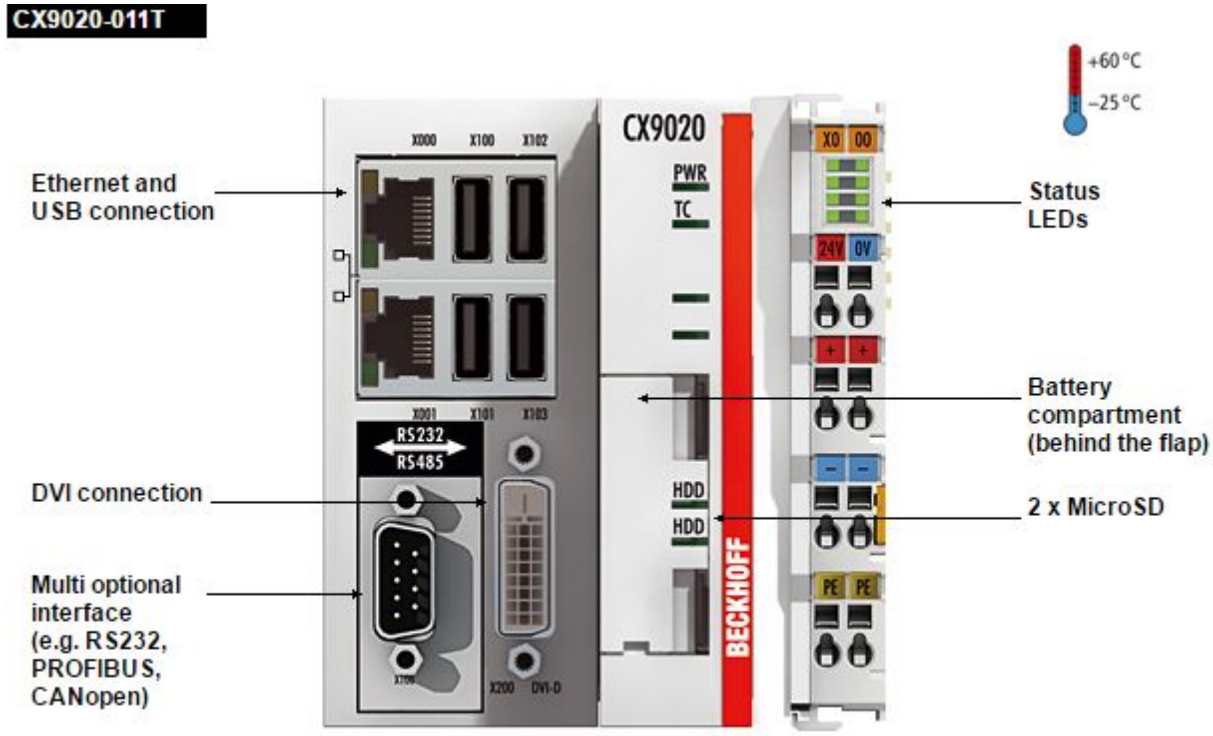

Figure 10. CX9020 Embedded PC Series. (Source: www.beckhoff.com , 2017)

There are a few options to connect the CX9020 to the clouds platform. First, it supports the OPC Unified Architecture communication protocol, which has tools available at *ThingWorx Marketplace* and in the *Azure IoT Gateway SDK* as third-party add-ons.

4.4.1 OPC Unified Architecture.

OPC Unified Architecture(OPC UA) is a Machine-to-machine (M2M) industrial protocol. It is a cross-platform service-oriented architecture (SOA) for process control. OPC UA supports two protocols, this is visible to application programmers only via changes to the URL. The binary protocol is **gh[Çl[hÆ××3]jn]j** and **g[hÇ`llhÆ××3]jn]j** is for Web Service (Streamed via XML files).

OPC UA is based on objects. An object that could be as simple as a single piece of data or as sophisticated as a process, a system or an entire plant. It might be a combination of data values, metadata and relationships. Take a dual loop controller. The dual loop controller object would relate variables for the setpoints and actual values for each loop. Those variables would reference other variables that contain meta-data like the temperature units, high and low setpoints and text descriptions. The object might also make available subscriptions to get notifications on changes to the data values or the metadata for that data value. A client accessing that one object can get as little data as it wants (single data value) or an extremely rich set of information that describes that controller and its operation in great detail.

An OPC UA server models data, information, processes and systems as objects and presents those objects to clients in ways that are useful to vastly different types of client applications. Better yet, the OPC UA server provides sophisticated services that the client can use, including:

Discovery Services – Services that clients can use to know what objects are available, how they are linked to other objects, what kind of data and what type is available, and what metadata is available that can be used to organize, classify and describe those objects and values.

Subscription Services – Services that the clients can use to identify what kind of data is available for notifications. Services that clients can use to decide how little, how much and when they wish to be notified about changes, not only to data values but to the meta-data and structure of objects.

Query Services – Services that deliver bulk data to a client, like historical data for a data value.

Node Services – Services that clients can use to create, delete and modify the structure of the data maintained by the server.

Method Services – Services that the clients can use to make function calls associated with objects.

Opc Ua Client - Is the side that initiates the communication session. Clients in OPC UA are flexible. It's capabilities include search and discover OPC UA servers, discover how to communicate with the server, configure the server to deliver specific pieces of data when and how they want it. Which means that OPC UA clients will generally support many different protocols mappings so that they can communicate with all different types of servers.

Opc Ua Server – An OPC UA server endpoint is the side of an OPC UA communication that provides data to an OPC UA client. There is no standard OPC UA server either in functionality, performance or device type. Devices from small sensors to massive chillers may be OPC UA servers. Some servers may host just a couple of data points. Others might have thousands. Some OPC UA servers may use mappings with high security and lower performance XML, while others may communicate without security using high performance OPC UA Binary Encoding. Some servers may be completely configurable and offer the client the option to configure data model views, alarms and events. Others may be completely fixed.

BLOB (Binary Large Object Block) – "BLOBs" provide a way to transfer data that has no OPC UA data definition. Normally, all OPC UA data is referenced by some sort of data definition which explains the data format. BLOB data is used when the application wishes to transfer data that has no OPC UA definition. BLOB data is user defined and could be anything: video, audio, data files or anything else.

Encodings – A data encoding is a specific a way to convert an OPC request or response into a stream of bytes for transmission. Two encodings are currently supported in OPC UA: OPC UA Binary and XML. OPC UA Binary is a much more compact encoding with smaller messages, less buffer space and better performance. XML is a more generic encoding that is used in many enterprise systems. XML is easier for enterprise servers to process, but requires more processing power, larger messages and more buffer space.

Security Protocol – OPC UA uses the same type of security used on the Internet for privacy and security: certificates.

Transports – A transport is the mechanism that moves an OPC UA message between a client and server. This is another term that at first glance can be confusing. All OPC UA messages are delivered over a TCP/IP connection. Within TCP, there is what I would call a session, though the word "session" is not specifically used anywhere that I've found in my study of the technology. There are two kinds of these sessions that message over TCP, and they are called transports when using OPC UA. They are OPC UA TCP and SOAP/HTTP.

4.4.2 Connection problems.

Beckhoff provides a software called "TS6100 - TwinCAT OPC UA Server" that can be installed in a *Windows* machine, even in the "*Compact Edition"* that runs in the CX9020. In order to be accessed by the third-party *OPC UA Client* extension in *ThingWorx*, accordingly to the user guide, a *jar* (*java archive*) and a *json* configuration files need to be put in the same folder, with administrative privileges, as the OPC UA server. This turned out to be a major problem, since many version of *Windows*, like the *Microsoft Windows Embedded Compact 7* running in the PLC, does not support *Java* applications and/or give administrative privileges to those. With this in mind, the connection to *ThingWorx* turned out to be a problem.

When changing to *Azure*, another obstacles showed up. After some meetings and email exchanges with engineers of *Beckhoff,* the connection solution presented was install the *TwinCAT OPC UA Server* in one virtual machine running inside *Azure*, and that machine should uses it own solution to route and store the data, since ,this way, only the access and connection to the *OPC UA* variables would be granted. This was not the best solution, configuring one database and data handling is onerous as implementing one gateway to the *IoT Hub*, which already handle these parts.

Azure IoT Gateway SDK is so new that its repository in *Github* is updated almost daily (when writing this, 25th of January, 2017, the last modification seen on the gateway's hub was 14 hours ago). Unfortunately, the third-party modules, like the OPC UA module (https://github.com/Azure/iot-gateway-opc-ua), are not up-to-date with the gateway, which means a lot of failures when trying to build the solution.

Some alternatives to remedy the situation were found, but not for free. One was to get data from the PLC using HTTP without encryption, and use one PHP script to receive the data and store it in one *txt* file. Not the most elegant and viable solution in one real business case scenario, also this way the connection would be one way only.

The last try was to use a gateway to go from ADS protocol (PLC native) to MQTT (*Azure* native). MQTT is a client-server messaging transport protocol which uses the pattern publish-subscribe[8] (Pub/Sub). It is designed to be lightweight, open, simple and easy to implement. These characteristics, make it ideal for use in applications like Machine to Machine communication (M2M) and IoT, where

[8] In software architecture, publish-subscribe is a messaging pattern where the sender of messages, called publishers, do not program the messages to be sent directly to specific receivers, but instead characterize published messages into classes. In this way the receivers, called subscribers, express interest in one or more classes and only receive messages that are of interest. Both publishers and subscribers are not aware of each other.

small code footprint is required.This is made by using one *C#/Windows Form Application* that reads data from the *TwinCAT Runtime* of the PLC.

Both of these previous examples can be found in the following *Github* repository:

- *IoT-Solutions* (2017): https://github.com/ldmontibeller/IoT-Solutions

5. Analysis and interpretation of results

All things considered, it is clear that Internet of Things still a challenge. There are some main reasons why, nowadays, IoT solutions are not plug-and-play. The first thing noticed is the presence of many legacy devices. Made before the connectivity era, these devices will almost always need one interface (gateway) in the middle of the path of data. Doesn't matter if the middle interface is a virtual machine in the cloud or a real physical device, some kind of "translator" will have to act to make cloud and device communicate.

This show the importance of the gateway. The second factor that makes IoT complex is the vast variety of communication protocols with different purposes. Some work on the premises of being scalable, like OPC UA, to use the same protocol in the edge and the cloud. This requires a well specified structure of data and metadata, which makes the protocol complex and sometimes hard to understand. At least is possible to see two patterns: the transport layer protocol adopted is usually TCP/IP and MQTT protocol seems to be the one that will be adopted for the most of the IoT platforms.

The gap between the engineering and IT is very visible. For me, as an engineering student, was clear that the cloud platforms are made and focused on web and, understanding terms and concepts of computer science were what took me most of the time. Making the knowledge from the industry floor engineer merge with the web-based software developer may be one of the great tasks in this field.

Another crucial thing is, as a business, IoT has a enormous value. However, how one company model they're IoT business may be decisive to the customer choose one over another IoT platform. After studying *ThingWorx* and *Azure* platforms, is notable how more closed is *Thingworx.* Is hard find support and help about the platform, the development using the SDKs is a bit hard because most of things are not open-source. There are some advantages over *Azure,* is really easy to deploy the solution and there are a lot of pre-made tools and extensions that only requires configuration. *Azure,* in comparison, has the *IoT Gateway SDK* and the modules on *Github,*which makes easier to understand how the tools work and makes it more flexible for different usages. Is way more easier to find material to learn about their platform, the IoT concepts and the concepts that can be found at *Microsoft Virtual Academy*. In the other hand, some documentation and material change so fast that, sometimes, is hard to deploy the solutions.

6. Final Considerations

Since the beginning of the project, was evident the improvement of knowledge in the subject field, both in theoretical and execution levels. I've understood what is the role of Internet of Things, how the data actually flows through it, at what step the current technologies are and what are the real practices of companies when connecting devices to the cloud.

My participation in this exchange program was also valuable for the finnish university. As I came from the automation field, it was possible to contribute with notions of industry floor level, likewise the contrast between brazilian and european automation industry.

When it comes to technical knowledge, many subjects can be cited: architecture and guidelines of Industrial Internet of Things; Web services; *HTTP* and *HTTPS*; *WebSockets*; notions of Data Science; development of *Windows* and *Linux* applications; *Python, C, C#* and *.NET* development; OPC UA and MQTT protocols; usage of *ThingWorx* solutions (*Composer, KEPServer EX, SDKs* and extensions); usage *Microsoft Azure* solutions (*IoT HUB, IoT Suite, IoT Gateway SDK, Windows 10 IoT Core*); usage of *Beckhoff* solutions (PLC programming with *TwinCAT 3, TC3 OPC UA Server*).

The experience of the exchange program cannot be measured. It is life-changing experience, valuable to the point that it changes mindsets and convictions. of the participant. I am thankful to the *PROPICIE* initiative, the *Federal Institute of Santa Catarina - IFSC* and to the *Hämeen University of Applied Sciences - HAMK*. I hope programs like that can be maintained and I dispose myself to make it happen so more students can use of the enriching cultural exchange experience.

Leonardo Drews Montibeller

7. Further Development

For whom it may concern to continue this work, I've suggest further study and development of the alternative solutions present in the *IoT-Solutions* repository, focusing also in the *IoT Gateway SDK* and the *MQTT* protocol, since these topics has been shown as the most promising.

8. REFERENCES

INDUSTRIAL INTERNET CONSORTIUM. **Industrial Internet Reference Architecture.** Version 1.7. 2015. Available at: <https://www.iiconsortium.org/IIRA-1-7-ajs.pdf> Jan. 11th, 2017.

NATIONAL INFORMATION STANDARDS ORGANIZATION**. Understanding Metadata.** NISO Press. 2004. ISBN: 1-880124-62-9. Available at <http://www.niso.org/publications/press/UnderstandingMetadata.pdf> Jan. 11th, 2017.

LITTLEFIELD, Matthew. **IIoT And Big Data Analytics: How Manufacturing System Architecture Is Being Transformed.** LNS Research. 2016. Available at <http://blog.lnsresearch.com/iiot_and_big_data_analytics__transforming_manufacturing_system_architecture> Jan. 12th, 2017.

CHOPRA, Varun. **WebSockets Essentials - Building Apps with HTML5 WebSockets.** Packt Publishing. 2015. ISBN-13: 978-1-78439-675-6.

KEPWARE TECHNOLOGIES. **KEPServerEX Product Overview**. 2017. Available at: <https://www.kepware.com/en-us/products/kepserverex/> Jan. 18th, 2017.

BROWN, Pete. **Getting Started with the Internet of Things (IoT)**. 2016. Available at: <https://mva.microsoft.com/en-US/training-courses/getting-started-with-the-internet-of-things-iot-16170>

STREET, Chipalo. **Get started with the Azure IoT Gateway SDK (Windows)**. 2016. Available at: < https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-windows-gateway-sdk-get-starteds>

MAHNKE, Wolfgang. LEITNER, Stefan-Helmut. DAMM, Matthias. **OPC unified architecture.** Springer Verlag. 2009. eISBN-13: 9783540688990.

PTC INC. **ThingWorx OPC UA Client Extension v1.1 User Guide**. 2016.

9. APPENDICES

Appendix I

What is a HTTP and REST API?

Leonardo Drews Montibeller
29/11/2016

HTTP stands for Hypertext transfer Protocol, is the foundation for the World Wide Web. It uses logical link between nodes containing texts. It works as a **request-response** protocol with connection between **client-server**, a web browser may be a client and an application running on a computer may be the server, containing **resources**, like **HTML** files. HTTP is an application layer protocol, and works on top of a transport layer protocol, usually TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).

HTTP resources are identified and located on networks by **URLs** (Uniform Resource Locators), using the **URIs** (Uniform Resource Identifiers) schemes HTTP and HTTPS. URIs and hyperlinks in one HTML document form inter-linked hypertext documents.

A web server is a just a computer program that listens for requests from browsers and then execute them. These requests and responses are issued in a special language called HTTP. It's worth noting that HTTP only defines what the browser and web server say to each other, not how they communicate. The actual work of moving bits and bytes back and forth across the network is done by TCP and IP, which are also used by FTP and WebSockets.

Most URLs have this basic form: "protocol://server/request-URI". The protocol part describes how to tell the server which document the you want and how to retrieve it. The server part tells the browser which server to contact, and the request-URI is the name used by the web server to identify the document.

The structure of a HTTP client request is like the following:

ß-%4(/$à ß2%15%34È52)à (440×ß6%2à
ß^a]d\fYe]úàÆ ß^a]d\ÈnYdm]úà these are called headers, or parameters of the request.

You can actually play web browser yourself and write HTTP requests directly to web servers. This can be done by telnetting to port 80 (default HTTP), hitting enter twice and typing the request, for example:

l]df]l]fÇoacah]\aYÇgj_ āù
And then
'%4 ×oaca×-Yafë0Y_] (440×úÇú
Or maybe
(%!$ × (440×úÇú

For HTTPS, which uses SSL certification you can use on a Linux machine the **gh]fkkdÆ**

gh]fkkd kë[da]fl È[gff][l \]nÈ`YecÇ]dakYaglÇ[geÆýýü

Watch the ssl certificate details scroll by and then
'%4 ×4`af_ogjp×4`af_k× (440×úÇú
(gklÆ \]nÈ`YecÇ]dakYaglÇ[ge
Yhh+]qÆ ppppÈ**pppp**È**pppp** (use your own application key)
pÈl`af_ogjpÈk]kkagfÆ ljm]
Hit enter twice and see the response.

For further reference you can check the HTTP documentation for all the available methods and what they do.

So, to do the same thing,in one browser address like this:

`llhkÆ××\]nÈ`YecÇ]dakYaglÇ[ge×4`af_ogjp×4`af_k×ËYhh+]qđáppppÈppppÈppppâµpÈl`af_ogjpÈk]kkagfđáljm]â

The default method in one browser is **'%4**, so everytime you type one address the browser is adding the **'%4** method to the string value.

`llhkÆ×× is the way, and the way for https, by default is the port 443

\]nÈ`YecÇ]dakYaglÇ[ge×4`af_ogjp×4`af_k× is the hostname of the server to be resolved

ËYhh+]qđáppppÈppppÈppppâµpÈl`af_ogjpÈk]kkagfđáljm]â
Everything after **Ë** are parameters (or headers, they are the fieldname and fieldvalue from the HTTP structure) that you can use knowing the your API, in this example we are using ThingWorx API, so it need a application key **ÝYhh+]qÞ** with the value of **áppppÈppppÈppppâ** and **ÝµÞ** the parameter **pÈl`af_ogjpÈk]kkagf** with value **áljm]â** to run the session without asking for credentials.

When the server receives the HTTP request it locates the appropriate document and returns it. However, an HTTP response is required to have a particular form. It must look like this:

(440×ß6%2à ß#/$%à ß4%84à
&a]d\úÆ 6Ydm]ú
&a]d\ûÆ 6Ydm]û

ÇÇÇ$g[me]fl [gfl]fl `]j]...

The first line shows the HTTP version used, followed by a three-digit number (the HTTP status code) and a reason phrase meant for humans. Usually the code is 200 (which basically means that all is well) and the phrase "OK". The first line is followed by some lines called the header, which contains information about the document. The header ends with a blank line, followed by the document content. Below, is a typical header:

(440×úÇù ûùù /+
3]jn]jÆ .]lk[Yh]È#geemfa[Ylagfk×úÇú
$Yl]Æ 4m]k\YqÄ ûþÈ.gnÈĂĀ ùúÆûûÆùý '-4
,YklÈeg\a^a]\Æ 4`mjk\YqÄ ûùÈ.gnÈĂĀ úùÆýýÆþü '-4
#gfl]flÈd]f_l`Æ ÿüĀû
#gfl]flÈlqh]Æ l]pl×`led

ĒÉ$/#490% (4-, 05",)# íÈ××7ü#××4 (4-, üÇû &afYd××%.íē
Ē(4-,ē
ÇÇÇ^gddgo]\ Zq \g[me]fl [gfl]fl...

Now that the fundamentals of how the world wide web works were presented, is time to understand the usage of API in web services. API (Application Programming Interface) is something that allows one piece of software to talk to another. And REST stands for REpresentational State Transfer.

Usually a REST API works like a website: a call from a Client to a Server via HTTP protocol. One main difference is the form that the information is represented. If you go to Google Maps API (http://maps.google.com/maps/api/geocode/json), you will get a message that some parameters are missing, so try http://maps.google.com/maps/api/geocode/json?address=hameenlinna&sensor=false. Then you got data about Hª meenlinna city in a JSON format (http://www.json.org/) .

References:

GARSHOL, Lars M. **How the web works: HTTP and CGI explained.** 1999. Available at http://www.garshol.priv.no/download/text/http-tut.html

WIKIPEDIA. **Hypertext Transfer Protocol.** 2016. Available at https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

NUZUM, Matthew. **How to use telnet to test ssl/https websites.** 2008. Available at https://www.bearfruit.org/2008/04/17/telnet-for-testing-ssl-https-websites/