Noname manuscript No.
(will be inserted by the editor)

A Full Stack for Quick Prototyping of IoT Solutions

Daniele Mazzei · Giacomo Baldi · Gabriele Montelisciani · Gualtiero Fantoni

Received: date / Accepted: date

Abstract The paper presents a novel approach for prototyping interconnected products belonging to the Internet of Things context. The proposed solution aims at merging the benefits provided by monolithic vertical approaches (where all the IoT elements are pre-selected, from the hardware to the cloud) with those of horizontal solutions (that leave the freedom to select the single components and write the integration code). The proposed solution allows to speed up the prototyping process and lets the developers focus on coding the product behaviours rather than solving customization issues. The advantages of the proposed solution goes beyond the prototyping, as the prototype can be easily converted into an industrially viable solution. The paper ends with a real case application where the proposed stack is used for the development of an IoT unit that converts industrial refrigerators into smart connected systems.

1 Introduction

As technology advances, people and things are becoming more and more connected[1]. The Internet of Things (IoT)[2] and the machine-to-machine (M2M) communication[3] are some of the core paradigms that compose such scenario. The IoT paradigm is based on the connection and data exchange between physical devices and applications [4], linking real life entities with the virtual world [5][6].

D. Mazzei
Computer Science Department
University of Pisa.
Largo Bruno Pontecorvo, 3, 56127 Pisa PI, Italy Tel.: +39-050-2212779
E-mail: mazzei@di.unipi.it

G. Baldi, G. Montelisciani
Zerynth, New York, USA
E-mail: g.baldi@zerynth.com - g.montelisciani@zerynth.com

G. Fantoni
Department of Civil and Industrial
Engineering, University of Pisa, Italy
E-mail: g.fantoni@ing.unipi.it

IoT is growing in parallel with the Lean Startup paradigm[7], an innovative business design approach derived from the Lean Production Strategy[8]. As a result, we assist to a process where startups and innovators are reinventing and converting everyday products into “smart” or “connected” products: physical devices with a digital service at their heart [9]. As the number of such devices is growing exponentially, the personal, professional and economic benefits for the entire society are huge [10].

In particular, embedded devices based on low-power and low-cost micro-controllers are expected to trigger the IoT revolution allowing the “smartification” of “Things” without requiring a complete product redesign [11] and, even more important, with a minimum impact on the hardware cost of the devices.

However, the difference between *Things* and *Devices* is remarkable. According to Fremantle[12] *Things* are objects of our everyday life placed in our everyday environment, like a car, a fridge but also a house or city. *Devices* are sensors, actuators or tags endowed with a computational unit, and are usually part of a *Thing*. For example, a popular Smart *Thing* is the “Smart Fridge”, a common fridge coupled with temperature monitoring, goods tracking and compressor control sensors integrated in an electronic *Device* typically endowed with a WiFi network connection [13]. On the other side, in order to become a useful object, the Smart Fridge also requires to be coupled with a cloud service that enables the end user to manage and interface with the system through web or mobile interfaces (*services*).

2 IoT Architectures and Business Models

With the growth of smart IoT devices, two specific business models for IoT software architectures emerged: vertical and horizontal (Figure 1)[14]. In the vertical approach the IoT devices are typically sensors and actuators nodes connected to the Internet directly or through a local gateway. These elements are then linked to proprietary cloud-based services creating monolithic vertical offers. Such approach has a twofold advantage for the end-user: (i) it reduces (or eliminates) any compatibility issue among the different elements of the system, and (ii) it centralizes the support service to a unique provider. A clear downside concerns the full dependence on the vendor for any technical change, update or upgrade of the final solution.

This model is easier to be maintained and sold, so that it has been adopted by various IoT startups focused on B2C (Business-to-Customer) where the end-user of the solution coincides also with the final customer of the product.

On the other side, a horizontal model can boost growth and disruptive innovation guaranteeing high modularity, scalability and customizability. For this reason, the horizontal approach has been preferred in B2B (Business-to-Business) cases where the smart product is just designed by a company [14] and then developed by system integrators, hardware and ICT companies.

From the software point of view, telcos and big cloud companies opted for very wide horizontal cloud based offers that are adaptable to a wide range of applications and products. However, such offers typically lack of a real link with hardware *devices*. Indeed, this approach perfectly matches with the business model of cloud providers: addressing the users towards fully cloud-based pay per use platform as a service (PaaS) architecture where IoT nodes are considered as trivial remotely controlled devices aimed at generating as large as possible data streams.

Moreover, these solutions are often not suitable for the production of a customer oriented smart products (*Thing*) also because local intelligence and computation are mandatory for guaranteeing the implementation of proper smart behaviours. Indeed, smart products need to guarantee off-line usage and local data storage in order to maintain provided services active also in case of temporary network failure (e.i. control of a smart fridge, security alarm monitoring, etc.).

Concerning the hardware, an important role to speed up the development phase and meet the industrial-grade requirements is played by prototyping solutions aimed at reducing the entry gap in programming and managing embedded devices for IoT applications. These solutions can be grouped into two categories according to the adopted approach: (i) solutions running on a proprietary hardware and (ii) solutions running on generic hardware. Solutions tailored for proprietary hardware are for example Particle and Onion. Particle[1] provides a family of MCU (Micro Controller Unit) based boards that are ready to be connected to a proprietary cloud, allowing a quick development of smart product prototypes by using an Arduino similar C language. Onion[2] produces a MPU (Micro Processor Unit) based board with Wi-Fi connection. In this last case a tiny version of the Linux OS is executed in the MPU, allowing the development of IoT solutions by using high-level programming languages like Python, Ruby on Rails and Java.

[1] www.particle.io
[2] www.onion.io

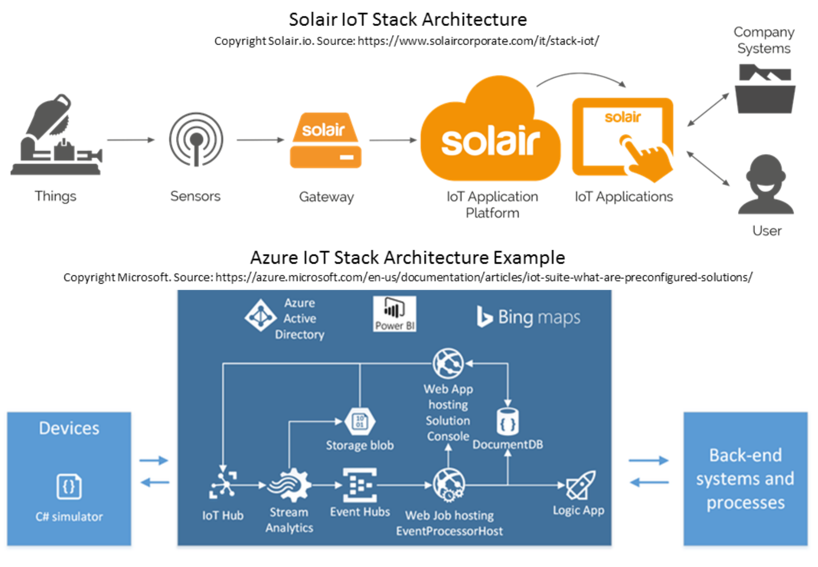

Solair IoT Stack Architecture
Copyright Solair.io. Source: https://www.solaircorporate.com/it/stack-iot/
Company Systems
Things
Sensors
solair
Gateway
solair
IoT Application Platform
solair
IoT Applications
User
Azure IoT Stack Architecture Example
Copyright Microsoft. Source: https://azure.microsoft.com/en-us/documentation/articles/iot-suite-what-are-preconfigured-solutions/
Azure Active Directory
Power BI
Bing maps
Web App hosting Solution Console
Storage blob
DocumentDB
Devices
C# simulator
IoT Hub
Stream Analytics
Event Hubs
Web Job hosting EventProcessorHost
Logic App
Back-end systems and processes

Fig. 1 Vertical and Horizontal IoT Architectures. On top the Solair.io vertical monolithic architecture that is typical of startups and SMEs, at the bottom an example of an IoT instance of the very wide horizontal Microsoft Azure cloud.

At MCU side, various software middle-wear able to run on various IOT hardware platforms, like Micropython[3] and Zerynth[4] VM (Formerly Viper VM), have been recently developed. Micropython is a Python interpreter executable by MCUs that allows a quick and easy development of embedded applications in Python language. Micropython scripts can be directly written on the board's serial port terminal giving developers the possibility to easily test embedded devices features and peripherals from a serial port emulated terminal console. The authors themselves have developed in 2014 VIPER (Viper is Python Embedded in Realtime)[15], a cross-platform Virtual Machine for MCUs that allows programming in Python language applications for MCU devices with real-time capability and with a very tiny footprint and low memory usage.

Within the described context, what is still missing to speed-up the development of smart products for IoT is a fully horizontal, modular, scalable and versatile IoT development stack. A system that can integrate the possibility to easily program, with high level languages, embedded devices that are natively connected to cloud based PaaS where data management modules, rule engines, data visualization tools and mobile App connection libraries are integrated and ready to use.

3 Technical specifications of a horizontal stack for rapid IoT prototyping

Hereinafter the problem is abstracted from real implementations and we try to identify the technical and architectural requirements of a horizontal stack for quick prototyping of IoT solutions and smart products. With reference to figure 2 going from bottom to top, and on the basis of the benchmark reported in [16], Technical Specifications **(T.S.)** and Design Requirements **(D.S.)** are here listed:

T.S. 0: Device Hardware layer. The stack has to be designed in order to guarantee an easy access to both boards (MCU and MPU) and sensors/actuators. In the case of quick prototyping every obstacle in choosing the hardware must be avoided. T.S.0 is more a prerequisite than a specification but constitutes the baseline for all the following T.Ss..

T.S. 1: Device Software layer. A multi-threaded Real-Time Operating System (RTOS) for embedded controllers allows users to easily manage the RT aspects of the device software application (embedded application), while a Virtual Machine or middlewear, that runs on top of the RTOS, allows users to program MCUs and MPUs in high level scripting languages (like Python, Javascript, Lua, etc.) thus increasing the re-usability of the code. The introduction of an high-level programming language doesn't have to interfere with the real-time and low-power capability of the system that are mandatory for industrial and professional IOT scenarios. In case of use of an MPU based board, the Linux OS can be installed allowing the use of high level languages. However, MPUs are not suitable for battery powered smart product due their intensive power consumption. Moreover, real-time features can't be always guaranteed by MPUs. For this reason, hereinafter the paper will focus on MCUs.

T.S. 2: Device network layer. This layer enables the co-existence of both hardware and software elements. Devices connectivity to the network is provided

[3] www.micropython.org
[4] www.zerynth.com

through network hardware units (WiFi, LAN, Cellular, etc.) and their related drivers. At a practical level the device network layer must support TCP and UDP connections and higher level protocols such as HTTP and HTTPS. Moreover, messaging protocols such as MQTT, CoAP etc. need also to be supported to allow a quick connection with third party clouds. Finally, this is the layer where other non-internet connections like the recently developed Low Power Wide Area Network (LPWAN)[17] (e.g. LoRa and Sigfox) are managed.

T.S. 3: Cloud network layer. The devices have to be easily manageable and the gathered data organized and stored on the cloud. Therefore a cloud based device manager, a data collection engine and a data storage must be available. This layer need to be easily customizable to allow the creation of data pipelines that will feed the higher levels of the cloud infrastructure.

T.S. 4: Cloud application layer. The real cloud applications and logic are implemented in this layer. Here engines for data mining, rule execution and data context analysis need to be instantiated. This layer need to be customizable by using high-level languages allowing an easy and quick definition of the cloud applications, logics and services.

T.S. 5: Business Intelligence layer. The end users finally need an effective way to see the data and to analyse them. A set of tools for the acquisition and transformation of raw data into meaningful and useful information is usually adopted to quickly provide synthetic information and *ad hoc* reports. Moreover, in a higher and higher connected world, a mobile App for easily discover, connecting and control all the IOT devices is mandatory.

D.R. 1: Developer friendly. From a developer point of view, a cross-platform Integrated Development Enviroment (IDE) with the possibility of managing all the above mentioned Device's layers in a "plug and play" approach would allow developers to use their own working station without spending time with the installing and configuring of multiple dedicated tool-chains (compilers, IDEs, and board specific tools).

D.R. 2: Mass production scalability. A quick prototyping software tool for embedded systems should enable to switch from the prototype to the final industrial version as well as easily change the prototyping hardware with a production oriented solution. Indeed, the system needs to include batch production tools like batch programming of devices and production analytics.

D.R. 3: Secure and Reliable by Design. A product-oriented development framework needs to guarantee the design of smart devices that will be secure and reliable by definition. Elements and infrastructures aimed at guarantee tamper and hacking resistance[18] can enable the devices' security since the design and prototyping phases. Moreover, being the design process oriented to the mass production, the framework has to ensure the tracking of the produced devices in order to avoid frauds at the production. In order to guarantee high security and reliability the system need also to be compliant with real-time specification and support low power features that are both mandatory in industrial, professional and commercial IOT scenarios.

D.R. 4: Native support for mobile applications In order to be defined as "Smart", a product need to be coupled with a multi-platform mobile App that allows users to easily interface with the devices, thus enabling different services and actions [9]. The coding of such App should be based on multi-platform languages,

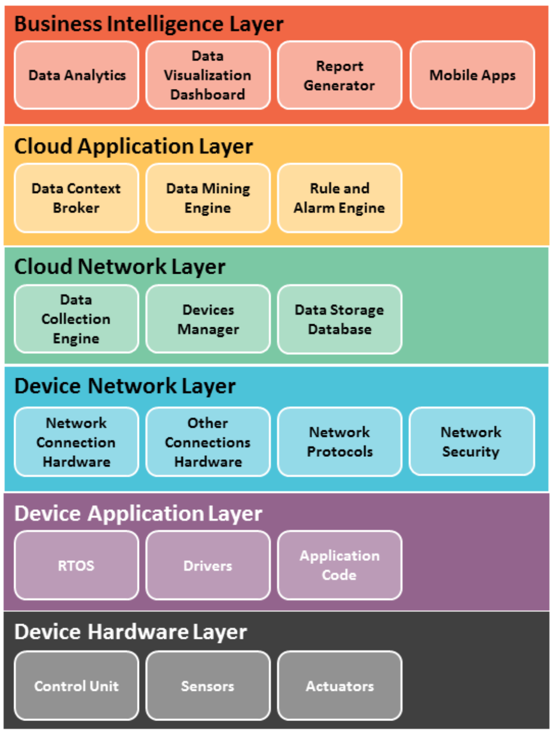


Fig. 2 Typical IoT Architecture. Data are collected at device level by means of sensors, streamed to the network by using different hardware and protocols while locally elaborated by the MCU thanks to the embedded application. On the Cloud side, data are stored, analyzed and reported by means of dedicated business intelligence engines and reporting interfaces.

avoiding the developers to re-implement the same App for each mobile platform available on the market.

4 The Zerynth Stack

This section introduces a full stack component solution for embedded devices (MCU only) going from cross-platform programming of the firmware to cloud data visualization, analytics and mobile integration. This architecture has been designed to be as modular and flexible as possible by allowing (i) a significant freedom in the choice of MCU based boards, sensors, actuators, real-time operating systems, and (ii) a seamless integration with third-party cloud based PaaS [15].

The presented architecture has been designed in order to become the core of the Zerynth solution that is represented in Figure 3. In this section the various components of the Zerynth Stack are described highlighting their links with the above mentioned Technical Specifications **(T.S.)** and Design Requirements **(D.S.)**:

Internet Connected Devices are physical devices based on MCU that can be programmed with Zerynth in a cross platform manner using the Python pro-

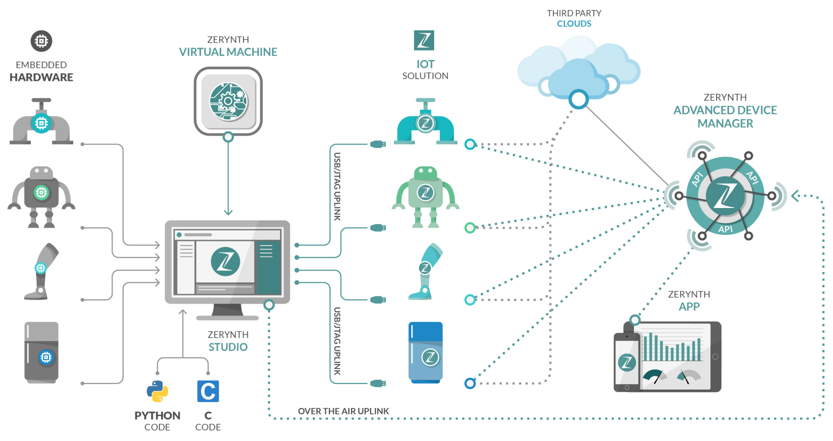


Fig. 3 The Zerynth Stack

gramming language. These devices are typically endowed with sensors and actuators that can be easily accessed and managed using the high-level Python libraries provided by Zerynth (**T.S.0** and **T.S.1**). Zerynth also supports TCP and UDP connections, allowing the use of HTTP and HTTPS protocols if the specific microcontroller has enough resources (memory) to run cryptography. Moreover, various network high-level communication protocols like MQTT and COaP are also supported and used as transport technology for the connection to third party clouds and PaaS. Zerynth also provides a generic cloud connector library written in Python that can abstract the details of connecting to a particular cloud service by exposing a unique cloud interface (Microsoft Azure, Amazon Web Services, IBM Bluemix, Google IOT and Ubidots are already supported) (**T.S.2**).

Non-Internet Connected Devices are physical devices that are not directly connected to the Internet via LAN or WiFi as they are endowed with other data connections like: ZigBee, LoRa, Sigfox, Z-Wave and similars. The Zerynth socket library allows data exchange using high-level protocols regardless the underneath connection. For this reason, non-Internet connected devices can stream and receive data using the same approach used by the Internet connected objects (**T.S.0** and **T.S.1**). Moreover, a Zerynth Internet connected device can act as a gateway for non-Internet connected devices acting as a bridge between the Internet and the non-Internet networks(**T.S.2**).

Zerynth Virtual Machine is a real-time operating system for microcontrollers coupled with a set of software layers aimed at executing applications programmed in Python and in C on the embedded device's MCU. The Zerynth VM supports multi-threads, exceptions and many other features typical of high-level programming languages. However, the RTOS also allow fast and time constrained routines to be executed.These features together with the cross-platform compatibility allow the reuse of code that can run on a wide range of different devices without modifications. Moreover, by using the Python language running on a Zerynth VM, the firmware programmer can focus more on the firware behaviours than on the complexities of the embedded low level code development (**T.S.1**).

Zerynth Advanced Device Manager is the cloud frontend towards the connected devices. The Zerynth Device Manager (ZADM) allows the management of devices by means of persistent bidirectional TCP connections. The device manager is engineered to support thousands of opened connections both to gather data and to send commands to connected devices or group of devices. The ZADM is also configurable to integrate devices without Internet connection capabilities, that can access the ZADM via an IOT gateway (**T.S.3**). Devices communicate with the ZADM via TCP connections over a secure channel (TLS 2.1) exchanging messages in JSON format. To ease the discovery and deployment phase, each device identifies itself with the ZADM using a one-time security token derived from the device micro-controller unique identifier. Depending on the desired deployment flow, the device can store a client TLS certificate to authenticate itself to the ZADM upon the first connection or can receive such certificate in exchange of the one-time security token. The choice of the deployment protocol depends by security requirements and by type of hosting (described below) used for the ZADM.

Through the Zerynth ADM it is possible to discover Zerynth connected objects with various levels of security and credentials associated with specific users or groups of them. The ZADM also provides to connected devices a periodic update that allows all the embedded units to be synchronized with the ZADM timestamp. Once received, the ZADM timestamp is used to set the real-time clock of the devices MCU and then used for correcting clock skewing and time tagging of the outcoming messages.

The Zerynth ADM can run on any Virtual Private Servers (VPS) service through Docker[5], allowing Over-the-Air updates of the Zerynth firmware (Bytecode) and/or of the Zerynth Virtual Machine. The ZADM has been designed in order to be deployed as a Docker container as well as a cloud microservice [19]. This allows an easy integration of the ZADM in cloud services where also security dedicated services like Blockchain [20] interfaces are located. In particular, the integration of the ZADM with a blockchain network would allow to certify the data collected and received by the devices making the IoT implemented architecture tamper proof and less vulnerable to hacking [21] [22] (**D.R.3**).

Device data received by the ZADM are then forwarded to third party clouds data stream processors using queue system like AMQP[6], RabbitMQ[7] or similar data ingestion protocols. This allow an easy integration with all the current available cloud based platform as a service architectures where event processors and data storage services are typically included.

Zerynth Studio is the integrated development environment (IDE) of the Zerynth Stack. It is a browser based IDE that runs on Windows, Linux and Mac. Through Zerynth Studio all the supported boards can be managed and the corresponding application developed in Python 3 language. Applications developed with Zerynth Studio can be saved locally or synced with users private and public GitHub repositories[8]. Zerynth Studio also includes a Package Manager (ZPM) for the easy installation and publication of libraries (**D.R.1**).

[5] www.docker.com
[6] https://www.amqp.org/
[7] https://www.rabbitmq.com/
[8] https://www.github.io/

Zerynth Uplinker is a component of the Zerynth Studio that allows uplinking Python applications onto the MCU via USB or JTAG connections. Over-the-Air uplink is also possible by means of the Zerynth ADM. Moreover, the Zerynth Uplinker is also available in a mass-production version that allows the flashing of Zerynth applications on multiple devices by using in-line programming facilities (**D.R.2**) (see figure 4). Moreover, since every Zerynth VM created is tied to the microcontroller unique identifiers (UID), Zerynth uplinker allows for firmware tampering protection (i.e. the VM will run only on the microcontroller with the specified UID) and for hardware and assets tracking, making the silicon vendor aware of the final use of the distributed microcontrollers (**D.R.3**).

Zerynth API Zerynth ADM services can be accessed via a generic outbound API to enable easy integration of external services. Push notifications and backend API for data monitoring and device control are one of the possible uses. The Zerynth API can be used for linking the Zerynth powered devices to other cloud services making the Zerynth Stack agnostic towards other SAS and APIs (**T.S.4** and **T.S.5**). The Zerynth API makes it possible to store devices' collected data on the preferred cloud or local database, or alternatively to push data into analytics pipelines and manage them through dedicated rule-engines.

Mobile SDK a particular instance of the Zerynth API, configured as a back-end API, is used in the Zerynth mobile SDK that allows fast development of mobile apps integrated with the Zerynth cloud (**D.R.4**).

Zerynth APP is a demo app based on a minimal set of features exposed by the Zerynth Mobile SDK. It is a ready to use App that acts as an interface for all the objects running Zerynth VMs. The Zerynth App just needs to be installed on a smartphone and/or tablet running Android or iOS. When launched it discovers all the Zerynth devices available on the local network or associated to the current user by means of the Zerynth ADM and Zerynth Mobile SDK. When one of them is selected by the user, the App becomes its interface. Zerynth App interfaces are based on HTML5 templates that users can select and edit through the Zerynth IDE and link them with specific functions of the Zerynth Python scripts. In this way there is no need to write any iOS or Android code. Device gathered data can be easily visualized on the Zerynth App HTML5 rendered interface while Zerynth device control function can be linked with the HTML5 interface buttons or other control elements. The Zerynth app also allows the triggering of iOS and Android native notifications that are shown on the user mobile device also if the App is closed (**D.R.4**). Moreover, templates developed for the Zerynth App can be also rendered in a common web-browser, thus empowering Zerynth devices also with a web interface.

4.1 Observations concerning the stack

It is worth noting that the bidirectional connection through the Zerynth ADM puts a lot of processing load when it deals with hundreds or thousands of devices. Such a choice could appear limiting when scaling. While having bidirectional persistent connections is the most powerful feature of the device manager, the "standard" connect-send-receive-disconnect pattern is also supported. The IoT device can decide the pattern. Persistent connections are easily managed with asynchronous

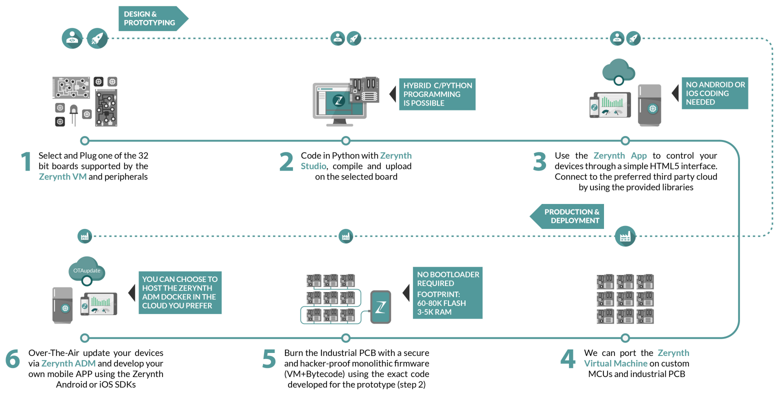

DESIGN & PROTOTYPING
HYBRID C/PYTHON PROGRAMMING IS POSSIBLE
NO ANDROID OR IOS CODING NEEDED
1 Select and Plug one of the 32 bit boards supported by the Zerynth VM and peripherals
2 Code in Python with Zerynth Studio, compile and upload on the selected board
3 Use the Zerynth App to control your devices through a simple HTML5 interface. Connect to the preferred third party cloud by using the provided libraries
PRODUCTION & DEPLOYMENT
OTAupdate
YOU CAN CHOOSE TO HOST THE ZERYNTH ADM DOCKER IN THE CLOUD YOU PREFER
NO BOOTLOADER REQUIRED
FOOTPRINT: 60-80K FLASH 3-5K RAM
6 Over-The-Air update your devices via Zerynth ADM and develop your own mobile APP using the Zerynth Android or iOS SDKs
5 Burn the Industrial PCB with a secure and hacker-proof monolithic firmware (VM+Bytecode) using the exact code developed for the prototype (step 2)
4 We can port the Zerynth Virtual Machine on custom MCUs and industrial PCB

Fig. 4 The Zerynth Design, Prototype and Deploy Workflow

frameworks (Zerynth is using Tornado[9] at the moment) up to many thousands. However, even if the objection of scalability is theoretically valid, the scalability problem in the number of connected devices has a less relevant practical implications. In real-world IoT scenarios with millions of devices in the field, probably the devices are fragmented in subsets, each one connecting to a different device manager node. Such implementation includes a load balancing system taking care of uniformly distributing the connections across available device managers. If more device management nodes are needed, they can be dynamically added.

Regarding the cloud application components, the stack does not impose any constraint on their placement. In particular, each component is shipped as a docker image that can be easily configured and executed both in a managed instance or in clustered mode for scalability purposes. This solution allows an easy and quick integration of the Zerynth stack in any third party cloud based platform as a service.

5 Use Case: the ROI Project

The Zerynth Stack has been instantiated for the implementation of the ROI (Refrigeration On Internet) smart product that represent a real-world scenario on which the versatility of the implemented stack is highlighted.

ROI is an easy to install and program monitoring and control unit for the refrigeration sector that can be installed on both old and modern commercial refrigeration systems and cooling units. ROI includes a end-user and a technician dashboard that allows data visualization and fridge control parameter setup together with a dedicated reporting section aimed at generating fridge statistics and normative reports. ROI project is a typical business case where the identification of a ready-to-use monolithic solution for the quick prototyping of the MVP (Minimum Viable Product) is very complex. This is mainly due to the need of a dedicated hardware unit able to monitor and interact with a wide range of different systems made of both old and modern refrigerators.

[9] http://www.tornadoweb.org

The ROI MVP set of features can be summarized in the following list: [**F.1**] A Device capable of reading sensor values and sending them through a wireless connection; [**F.2**] A Device Manager aimed at handling and monitoring the devices status; [**F.3**] A Context Broker aimed at managing the data packages sent by the devices; [**F.4**] An Event Processor aimed at controlling the data packages and eventually notify a non correct status of the devices; [**F.5**] A Database aimed at storing all the information coming from every component of the tool-chain that needs to save data; [**F.6**] A Data Viewer aimed at showing and reporting the data stored in the database in a user comprehensive way (such as charts, tables, graphs).

These features have been enabled by acquiring the following fridge data:

Tr	refrigerated room temperature (Degree Celsius);
Hr	refrigerated room humidity (Relative Humidity Percentage);
Tev	evaporator temperature (Degree Celsius);
Ten	environmental temperature (Degree Celsius);
Ds	door status (open/closed) (boolean);
Dt	door opening time (seconds);
Cs	compressor status (On/Off) (boolean).

5.1 Hardware and software Setup

5.1.1 Sensors, Connection Peripherals and Microcontroller board

ROI MVP electronic control board has been based on the Quail board produced by Mikroelektronika[10] that mounts a STM32 Cortex M4 MCU and has 4 mikroBUS sockets for the connection of external modules. The WiFi3 Mikroe Click based on the ESP8266 system on chip module has been used as WiFi connection module. ROI temperatures probes have been based on the DS18B20 digital temperature sensors that provide pre-calibrated 12-bit temperature readings over 1-Wire interface in the range $[-55°C - +125°C]$ with an accuracy of $\pm 0.5°C$. For the interfacing of the 1-Wire probes with the Quail board the I^2C 1-Wire click based on the DS2482-800 bridge has been used.

5.1.2 Embedded and Cloud Software

The entire ROI control unit embedded application has been programmed with only 300 lines of Python 3.4 using the Zerynth Studio and the Zerynth Virtual Machine All the drivers required by the external hardware modules and sensors were already available on the Zerynth Package Manager as free Python libraries. The ROI IoT Cloud architecture has been based on an instance of the Zerynth stack linked with various FIWARE Cloud[11] generic enablers.

The Zerynth ADM has been used for the management [**F.2**] of the various fridge monitoring units [**F.1**]. The control unit has been programmedin in order to send the Above mentioned fridge data every minute to the ZADM using a JSON packet.

[10] www.mikroe.com
[11] www.fiware.org

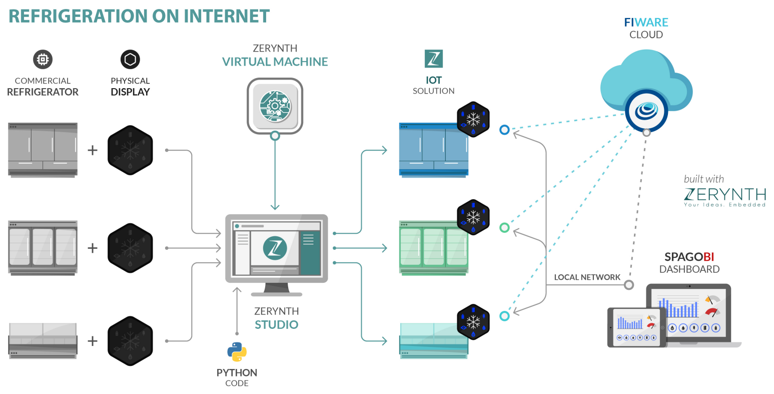


Fig. 5 The Zerynth Stack instantiated for the ROI Architecture

The **ORION** Context Broker FIWARE generic enabler has been used for the management of the devices data flow linking it to the Zerynth Advanced Device Manager via Zerynth API [**F.3**]. **CEP** Context Event Processor from Fiware has been used for the analysis of fridge's data in real-time and as events/alarms trigger. The ROI cloud architecture has been designed in order to be very versatile making it easy to adapt also to other industrial data gathering scenarios by means of a processing rule-set redesign [**F.4**]. **SPAGOBI** Business Intelligence engine has been used as data visualization engine and reporting interface [**F.6**]. Finally, data received from ZADM have been also stored in parallel on a MongoDb[12] database by means of a custom Python module connected to the ZADM via Zerynth API [**F.5**]. The above mentioned modules and software have been instantiated on a cloud architecture by encapsulating them in different Docker containers.

5.2 Test and Results

this section has been added during the review process

The ROI system has been installed in two pilot scenarios. The two pilot use cases have been selected in order to test the system in different commercial targets with different operational routines and environmental parameters. The first installation has been done on a meat transformation and supply company where various large (around 30-40 square meters) refrigerated rooms are used for the storage and the transformation of meat quarters. This installation allowed testing the system in an industrial environment where environmental condition are very aggressive and dynamic and where daily routine and operations can seriously stress the ROI hardware and setup. In this facility one device has been installed for the monitoring of a 30 square meteres refrigerated room. The second installation has been done on a typical Italian restaurant located on a very popular touristic location (Elba Island) where two ROI units have been installed for the monitoring of a fridge and a freezer dedicated to the storage of fresh local fish. In this context, refrigerators have been used with a daily routine that is strongly influenced by the weekly cycle

[12] www.mongodb.com

of touristic income but the environmental conditions are less aggressive than in the meat transformation facility.

The three installed devices has been left in place for 3 months and only few disconnections due to problem with the companies Wi-Fi networks have been recorded (see figure 6 where data on right side are missing). Alarms for over range temperatures have been correctly fired and notified via email to the companies' managers. The cloud system has been also configured for the generation of weekly temperature logs that have been formatted according with the HACCP (Hazard analysis and critical control points)[13] regulation guidelines (see figure 7) and used by the managers for their declarations.

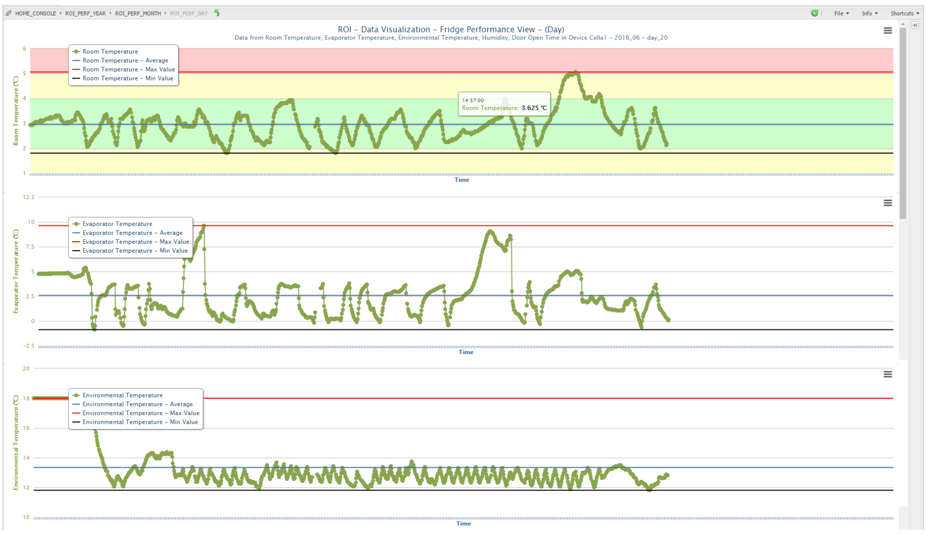
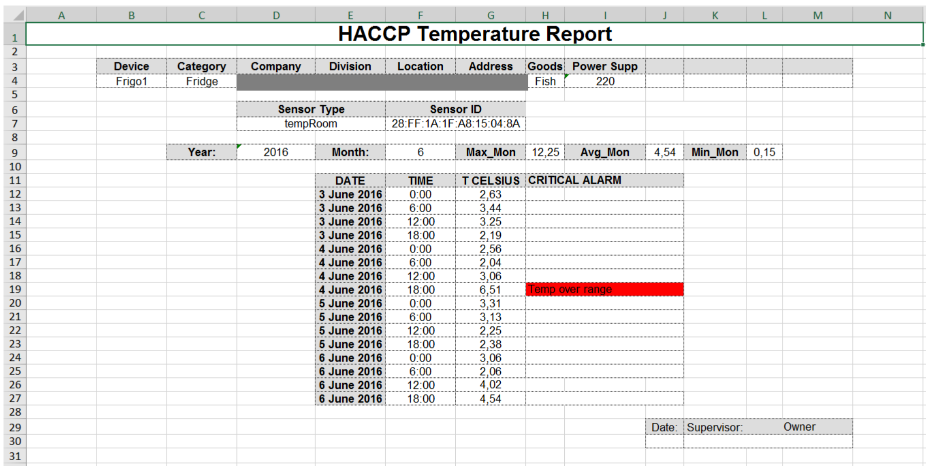


Fig. 6 The ROI fridge monitoring dashboard reporting data of the meat processing facility monitored room.

6 Conclusion

The paper presents an approach for the fast prototyping of IoT devices through a stack that enables the development process from the hardware layer to the cloud services. The advantages of programming with this approach include: (i) Coding in Python or Hybrid C/Python with a multi-threaded real-time OS that requires a footprint of just $60-80$k of Flash and $3-5$k of RAM; (ii) Development of flexible, scalable and customizable IoT solutions with reduced development time and high reliability; (iii) Cloud connection with the PaaS that best fit with technical technical and business requirements of the industrial scenario; (iv) Code reuse from prototype to industrial-grade solutions.
The solution has been successfully applied to a real industrial application: an IoT

[13] https://www.fda.gov/Food/GuidanceRegulation/HACCP/



HACCP Temperature Report

Device	Category	Company	Division	Location	Address	Goods	Power Supp
Frigo1	Fridge	[redacted]	[redacted]	[redacted]	[redacted]	Fish	220

Sensor Type	Sensor ID
tempRoom	28:FF:1A:1F:A8:15:04:8A

Year:	2016	Month:	6	Max_Mon	12,25	Avg_Mon	4,54	Min_Mon	0,15

DATE	TIME	T CELSIUS	CRITICAL ALARM
3 June 2016	0:00	2,63	
3 June 2016	6:00	3,44	
3 June 2016	12:00	3.25	
3 June 2016	18:00	2,19	
4 June 2016	0:00	2,56	
4 June 2016	6:00	2,04	
4 June 2016	12:00	3,06	
4 June 2016	18:00	6,51	Temp over range
5 June 2016	0:00	3,31	
5 June 2016	6:00	3,13	
5 June 2016	12:00	2,25	
5 June 2016	18:00	2,38	
6 June 2016	0:00	3,06	
6 June 2016	6:00	2,06	
6 June 2016	12:00	4,02	
6 June 2016	18:00	4,54	

Date:	Supervisor: Owner

Fig. 7 HACCP report generated by one of the devices installed at the restaurant.

monitoring system for industrial refrigeration presenting characteristics of modularity and scalability that could not be reached with the typical IoT platform described in section 2. Even if several elements of the stack have already been developed, future works will be oriented to (i) extend the number of supported RTOS, MCU architectures and boards, (ii) extend the device and cloud network facilities to low-range and many other communication protocols, (iii) improve the stack with a native connection to decentralized security technologies such as blockchains [23] and other cyber-security infrastructure, (iv) develop new sensors, actuator and industrial protocol interfacing libraries.

Sometimes, however the choice of the programming language mainly depends on the language the programmer prefers rather than on other parameters (reusability and readability of the code, short developing time, scalability of the solution, etc..). Conversely, enterprises are looking for tools able to reduce the development time and time-to-market and to make microcontrollers programmable by people with no competences in assembler and C.
Future works are oriented also to measure the differences (in term of time, reusability, etc..), if any, in the development of interconnected objects by using different approaches: Zerynth, MicroPython, Micrium, and compare them with the classical developments in C/C++ and assembly. At present, various universities and research centres are using Zerynth benchmarking it with respect to other approaches while several companies choose Zerynth as their development tool for IoT products and are providing important market feedback and new requirements. In our opinion, only such kinds of on-the-field measurements can provide a measurable evidence of the real advantages introduced by the presented technology confirming or refuting the presented hypotheses.

Acknowledgment

The authors thank the backers of the Kickstarter campaign who supported the development of the Viper VM, the 7000 active users of Zerynth and the INCENCe

Acceleration program (Grant N.632852) for supporting the development of Refrigeration on Internet project.

References

1. R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: the internet of things architecture, possible applications and key challenges," in *Frontiers of Information Technology (FIT), 2012 10th International Conference on.* IEEE, 2012, pp. 257–260.
2. L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
3. Y. Huang and G. Li, "Descriptive models for internet of things," in *Intelligent Control and Information Processing (ICICIP), 2010 International Conference on.* IEEE, 2010, pp. 483–486.
4. T. Fan and Y. Chen, "A scheme of data management in the internet of things," in *2010 2nd IEEE InternationalConference on Network Infrastructure and Digital Content.* IEEE, 2010, pp. 110–114.
5. Y. Huang and G. Li, "A semantic analysis for internet of things," in *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, vol. 1. IEEE, 2010, pp. 336–339.
6. D. Mazzei, G. Fantoni, G. Montelisciani, and G. Baldi, "Internet of things for designing smart objects," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on.* IEEE, 2014, pp. 293–297.
7. E. Ries, *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses.* Crown Books, 2011.
8. J. F. Krafcik, "Triumph of the lean production system," *MIT Sloan Management Review*, vol. 30, no. 1, p. 41, 1988.
9. C. Rowland, E. Goodman, M. Charlier, A. Light, and A. Lui, *Designing Connected Products: UX for the Consumer Internet of Things.* " O'Reilly Media, Inc.", 2015.
10. T. Yashiro, S. Kobayashi, N. Koshizuka, and K. Sakamura, "An internet of things (iot) architecture for embedded appliances," in *Humanitarian Technology Conference (R10-HTC), 2013 IEEE Region 10.* IEEE, 2013, pp. 314–319.
11. S. Krčo, B. Pokrić, and F. Carrez, "Designing iot architecture (s): A european perspective," in *Internet of Things (WF-IoT), 2014 IEEE World Forum On.* IEEE, 2014, pp. 79–84.
12. P. Fremantle, "A reference architecture for the internet of things," *WSO2 White Paper*, 2014.
13. J. Rouillard, "The pervasive fridge. a smart computer system against uneaten food loss," in *Seventh International Conference on Systems (ICONS2012)*, 2012, pp. pp–135.
14. R. Quinnell, "Vertical vs. horizontal: Which iot model will thrive?" http://www.embedded.com/electronics-blogs/other/4422131/Vertical-vs–horizontal–Which-IoT-model-will-thrive-, 2013, accessed: 2016-09-01.
15. D. Mazzei, G. Montelisciani, G. Baldi, and G. Fantoni, "Changing the programming paradigm for the embedded in the iot domain," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on.* IEEE, 2015, pp. 239–244.
16. J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of internet-of-things platforms," *Computer Communications*, vol. 89, pp. 5–16, 2016.
17. M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios," *arXiv preprint arXiv:1510.00620*, 2015.
18. T. Xu, J. B. Wendt, and M. Potkonjak, "Security of iot systems: Design challenges and opportunities," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design.* IEEE Press, 2014, pp. 417–423.
19. D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, 2014.
20. M. Swan, *Blockchain: Blueprint for a new economy.* O'Reilly Media, Inc., 2015.
21. K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
22. M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond bitcoin," *Applied Innovation*, vol. 2, pp. 6–10, 2016.
23. A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Security and Privacy (SP), 2016 IEEE Symposium on.* IEEE, 2016, pp. 839–858.