Towards an Easily Programmable IoT Framework Based on Microservices

Francisco Ortin[1*], Donna O'Shea[2]

[1] University of Oviedo, Computer Science Department, c/Calvo Sotelo s/n, 33007, Oviedo, Spain
[2] Cork Institute of Technology, Computer Science Department, Rossa Avenue, Bishopstown, Cork, Ireland

* Corresponding author. Tel.: +34 985 10 3172; email: ortin@uniovi.es
Manuscript submitted October 9, 2017; accepted December 1, 2017.
doi: 10.17706/jsw.13.2.90-102

Abstract: The number of devices connected to the Internet of Things (IoT) is increasing so rapidly that end-users with no programming background will demand the implementation of their own IoT services and applications. However, IoT programming is still a difficult task because of device heterogeneity, dynamic changes in the physical environment, and scalability, security, accessibility and availability issues. Many of these liabilities are also present in distributed systems, where microservice architectures are successfully used. Therefore, in this article we propose an IoT framework based on microservices to ease the development of IoT software. Visual programming is proposed to permit end-users to build simple services and applications. Visual dataflow abstractions declaratively identify the "things" and services in the network, creating a new level of indirection to create applications capable of adapting to changes in the IoT network. The devices connected to the network must provide a semantic self-description in order to support a global discovery service. End-users could describe the domain logic with existing visual programming abstractions previously proven to be suitable for non-programmers. The visual programs will be transparently compiled and deployed as microservices in a cloud-based environment, optimizing network traffic and runtime performance, while ensuring scalability, accessibility and availability. Software container technologies will be utilized to provide self-deployment of microservices.

Key words: Internet of things, visual programming language, microservices, cloud computing, software container.

1. Motivation

IoT refers to the networked interconnection of everyday objects, which are often equipped with ubiquitous intelligence [1]. IoT increases the ubiquity of the Internet by integrating every object for interaction, leading to a highly distributed network of devices communicating with human beings as well as other devices [2]. IoT applications are present in many different areas such as building and home automation, health care, smart cities, education, transport, manufacturing and environment monitoring [3]. Gartner forecasts that 8.4 billion "things" will be connected worldwide at the end of 2017, up 31 percent from 2016, and will reach 20.4 billion by 2020 [4].

IoT applications involve interactions among a set of different heterogeneous devices. Some of them may just interact with the physical environment (e.g., sensors and actuators), others may provide very specific services (e.g., data storage and social network interaction), and others allow programming new services (e.g., dedicated servers and cloud computing). This heterogeneity is not only present in the physical devices

but also in the big range of platforms, operating systems and programming languages used to create IoT applications. Besides this heterogeneity, other issues such as device discovery, security and scalability make the development of IoT applications a difficult task [5].

When developing an IoT application, developers build software using different types of physical devices for a particular purpose. Ideally, an IoT framework facilitates this task by providing high-level abstractions to orchestrate the services running in the heterogeneous devices. The approach of using high-level abstractions to express local device behavior was already addressed in the field of Wireless Sensor Networks (WSN) to facilitate application development and tolerance to failure [6], [7]. Another approach is based on *macroprogramming* or *large scale*, where the global behavior of distributed computation is declaratively specified over a WSN, hiding from the programmer details such as distributed-code generation, remote data access and management, and inter-node program flow coordination [8].

Although WSN is an important element of the IoT domain, IoT considers a greater range of devices [9]. Therefore, IoT applications commonly deal with different types of devices, platforms, operating systems and programming languages [10]. To deal with this high level of heterogeneity, model-driven [10] and generative framework approaches have emerged [11]. These systems allow the high-level macroprogramming of heterogeneous devices with different domain-specific programming languages to specify the different concerns involved in an IoT application [10]. Following this trend, we believe that the forthcoming growth of IoT will demand systems that allow the creation of IoT applications and services by end users with no programming background [12] (IoT services represent business activities provided as black boxes, allowing the interaction with the physical world and commonly consisting of other underlying services [13]; whereas IoT applications provide different IoT services to the final user requiring human interaction [14]).

Previous studies have identified visual programming languages as a suitable approach to build software by users with no programming background [15], [16]. If fact, there exist different visual programming languages and tools to develop programs for IoT devices [17]. We think visual programming could also be used to create global IoT services and applications by end-users, by visually orchestrating the existing services in an IoT network including those interacting with the physical environment through heterogeneous devices.

In this paper, we identify all the key elements to create an IoT framework to ease the development of IoT applications and services. The services provided by the physical devices, by the IoT framework and by the user are provided as microservices, taking advantage of the adaptability, heterogeneity, and fault tolerance benefits of the microservice architecture [18]. End users with no programming skills will be able to describe IoT applications and services with a visual language to declaratively specify how to orchestrate the existing IoT microservices for a specific purpose. They will also identify the “things” declaratively, allowing the development of IoT services and applications capable of adapting to physical changes in the network. Framework and user microservices (non-device microservices) will be deployed in a cloud-based environment, ensuring scalability, accessibility and availability. The proposed system will transparently move those microservices to select the most appropriate node to deploy them.

The rest of the paper is structured as follows. The following section describes the architecture of the proposed IoT framework and its key elements. Section 3 illustrates an example case scenario to show how the different elements of the proposed framework are used in a particular IoT application. Related work is detailed in Section 4, and Section 5 presents the conclusions and the future work.

2. Key Elements

We first identify the key elements required to build the proposed framework, where end-users could

visually program IoT applications and services. Figure 1 shows how these elements are related in the proposed architecture, based on the IoT World Forum Reference Model (WFRM) [19].

2.1. Microservice Architecture

A microservice is a small application with a single responsibility, which can be deployed, scaled and tested independently [20]. Microservice architecture is a particular way to design software orchestrating and choreographing these independently deployable services [18]. Common benefits of this architecture are decentralized control and data management, automated deployment, and evolutionary design [18]. These features make microservice architecture ideal for the proposed IoT framework. Its "smart endpoint and dump pipes" approach makes applications to be highly decoupled and cohesive, focusing on their own domain logic and acting as filters that receive a request, apply domain logic and produce a response [18]. The logic behind these services is commonly known by end users, who generally do not have any programming experience.

As shown in Fig. 1, the proposed IoT framework offers most of its functionality as microservices (layer 5) to develop IoT applications and services with all the benefits of a microservice architecture. First, physical devices such as sensors, actuators and computing nodes offer their functionalities as microservices. Second, other microservices abstract the different facilities provided by the IoT framework, such as device discovery (Section 2.2). Finally, the user may visually program their own domain-specific microservices (Section 2.5) and declarative queries (Section 2.3) to avoid the coupling between devices and applications.

The known benefits of microservice architecture [18] are particularly suitable in the IoT scenario. Its evolutionary design facilitates the replacement and upgrade of microservices, suitable to address the required adaptability to physical changes in IoT environments. The decentralized governance benefit allows the use of different technologies and programming languages for each microservice, supporting the existing heterogeneity in IoT applications. Its design for failure considers that any microservice could fail due to unavailability of the supplier, as happens in distributed IoT systems, providing more resilience to applications. The infrastructure automation benefit promotes scalability and automated deployment of those microservices not running in specific IoT devices (e.g., framework services such as device discovery and query, and user-defined services). Finally, the smart endpoint and dump pipes approach allows the creation of applications by end-users using orchestration and choreography tools. Since the logic of the smart endpoints are known by the final users, they could use visual abstractions to orchestrate the existing microservices for a particular purpose.

2.2. Dynamic Discovery of Self-Described Elements

The number of "things" connected to IoT networks increase rapidly. These "things" provide different interfaces and services. New devices are usually added to the network, and existing ones could be replaced and even removed. IoT applications must be able to adapt to these physical changes in the IoT networks. Therefore, an IoT framework should provide a mechanism to dynamically discover and integrate IoT devices and any other microservice [21].

In Figure 1, we identify the dynamic discovery of IoT elements as the third layer, above the physical devices and connectivity layers of the WFRM. Every time a new device is connected to the network, a dynamic self-description of the service(s) it provides should be included in the system. The numerous existing standards to describe devices (such as SensorML, OGC/SWE, W3C SSN and HyperCat) should be considered and generalized with a common scheme. Semantic-based technologies will play an important role in this context [22].

The user will be able to create new applications (layer 6) and services (layer 5) by using a visual programming language over the IoT framework (Section 2.5). After authentication, they will only be

allowed to access those devices they are authorized for. The new services created by users will also be included in the dynamic discovery service (`User` microservice in Fig. 1), available to create other applications and services. This dynamic discovery layer will use one of the existing approaches to discover microservices in a microservice architecture [23].

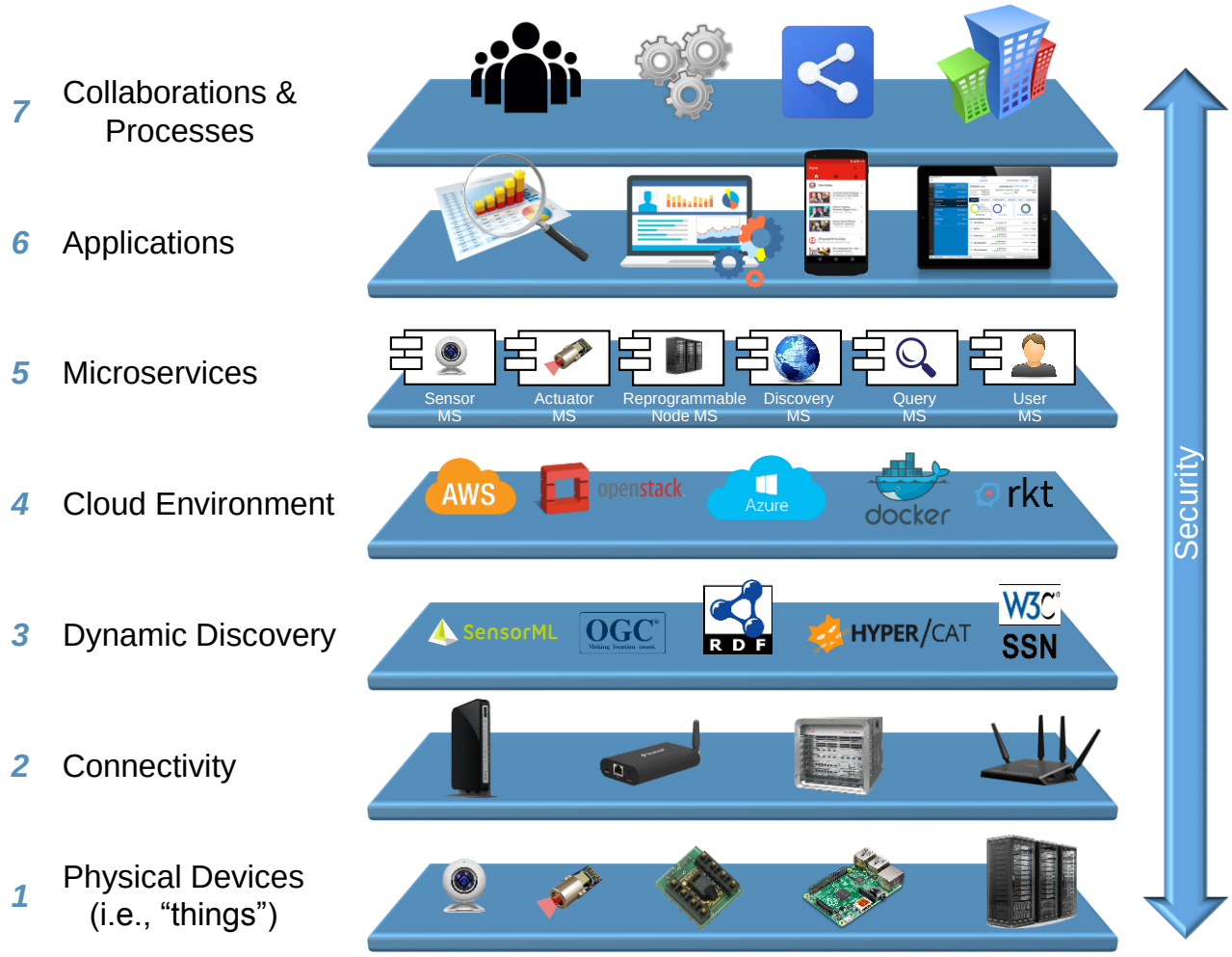

Fig. 1. Architecture of the proposed IoT framework.

2.3. Dynamic Discovery of Self-described Elements

As mentioned, the physical devices connected to the Internet are constantly changing. For instance, new sensors and actuators may be added to a building, some may be removed, and computing nodes may be reprogrammed. Therefore, the explicit access to physical devices implies high coupling and hence adaptability issues [24].

To avoid this coupling, we propose the identification of devices by means of declarative queries that select the them by dynamically checking their description, which may include sematic information (layer 3) [25]. For instance, instead of consulting the presence sensors 1, 2 and 3 in room A, we may just call a query microservice to know whether there is somebody in room A. Therefore, if new presence sensors are added to that room, or simply replaced with different ones, the application using the query will continue working. The proposed declarative access to physical devices represent a new abstraction level between the IoT devices and applications, permitting applications to adapt to changes in the physical environment.

These queries will be programmed by the end user using a visual dataflow language (an example will be

shown in Section 3). Its implementation is deployed as another microservice to be used by IoT applications. The microservice architecture allows applying this approach not only to devices, but also to any microservice (layer 5).

2.4. Microservice Deployment in a Cloud Environment

Those microservices providing the functionalities implemented by the physical devices (device microservices) will be running in the devices themselves. However, those providing functionalities of the IoT framework and those programmed by the users (non-device microservices) will be dynamically deployed to minimize network traffic and maximize runtime performance. For instance, the example query of presence sensors in a room would be ideally deployed close to the network where the sensors are connected to. The IoT framework will use its dynamic discovery microservice (Section 2.2) to locate the best node to deploy that microservice.

Besides choosing the most appropriate node, non-device microservices will be deployed to a cloud environment (layer 4). In this way, scalability, accessibility and availability are ensured [26]. The use of a high-level simple visual language also facilitates this task, since the language could be easily and transparently translated into different languages (e.g., Python, JavaScript and Ruby) and frameworks (e.g., Django, Node.js and Rails, respectively).

Microservices are developed in different languages, platforms, operating systems and frameworks, communicating with a lightweight mechanism (often an HTTP API) [18]. In a cloud-based environment, the deployment of these microservices require the configuration of cloud instances where microservices can run. Software containers (operating-system-level virtualization) allow the existence of multiple isolated user-space instances, and are easily packaged, lightweight and designed to run anywhere [27]. For this reason, the framework could use software containers such as Docker and CoreOS RKT to provide self-deployment of microservices [28].

2.5. Visual Programming

When end-users connect different devices to the Internet, they will probably be interested in building services to orchestrate those devices for some particular purpose. Due to the rapid growth of IoT, end users with no programming background will eventually demand the implementation of their own services [29]. Existing studies have identified visual programming as a suitable approach to build simple programs by non-programmers [15, 16]. For this reason, we propose the use of visual languages to allow non-programmers to build simple IoT microservices and applications, and the declarative queries identified in Section 2.3.

The users will have a visual mechanism to identify the existing microservices they are authorized to use. Utilizing these microservices, a domain-specific application, procedure or another service could be implemented with visual abstractions. The visual compiler will generate code using the appropriate technology in the server transparently selected for deployment (Section 2.4). The generated code could use orchestration and choreography technologies used in microservice architectures such as Activiti, Apache ODE, MQTT and RabbitMQ [23].

3. Example Case Scenario

In order to show how the proposed framework works, we describe an example use case. In this example, an end-user creates a service that connects to the web calendar of a company, selects the meeting rooms of one building, and warms up each room by turning on its heaters five minutes before each scheduled meeting.

Fig. 2 and 3 show mock-ups of a web front-end to visually program the framework. First, the user

authenticates himself or herself using any web browser. Then, the left-hand side of the system shows the different elements the user is authorized to use. Examples of those elements are the running services, queries, sensors, actuators, processing nodes, data sources, storage and programming elements. Except the last one, all of them are implemented as microservices (Fig. 1), but the programming environment classifies them differently, regarding its nature (Services, Queries, Sensors, etc.).

In order to implement the room warming service described above, the user first creates a declarative Presence query to know if there is somebody in one room (as explained later, the heaters will turn off when nobody is in the room). Fig. 2 shows this use case. We follow a dataflow approach similar to visual tools such as RapidMiner and Node-RED. The query being created has three inputs (country, place and roomID) to identify the room we want to warm up. Its output is a Boolean value indicating whether there is someone in that room.

The query implements a filter for the sensors the user has access to (i.e., the Sensors box). The first Filter box identifies those placed in the specified room, place and country (right-hand side of Fig. 2). The second Filter selects only those sensors aimed at presence detection. Finally, the Any higher order function returns one single value representing the presence of somebody in that room (true if any presence sensor detects that).

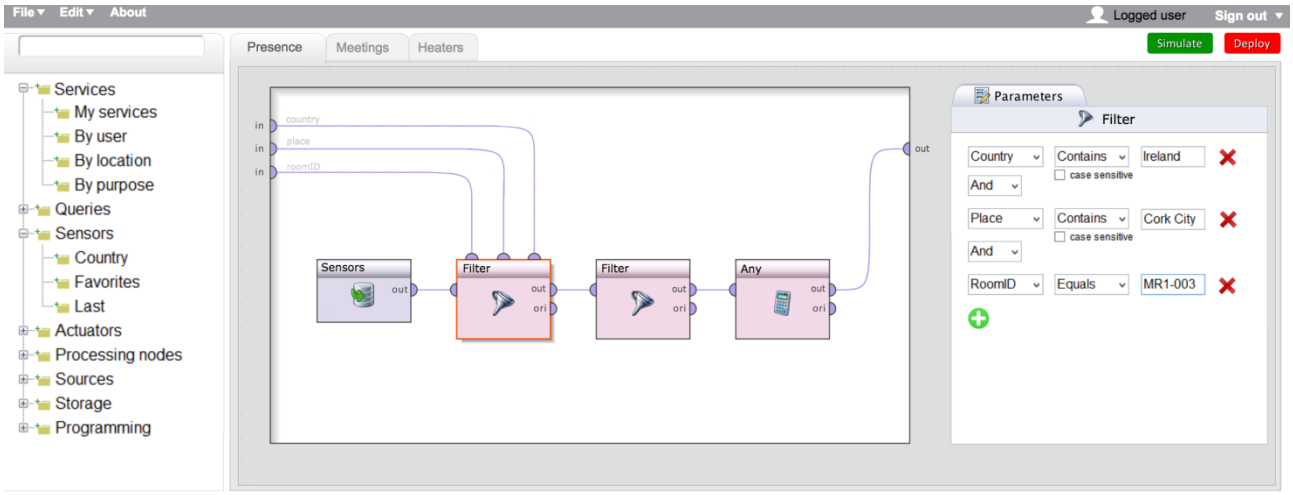

Fig. 2. Creation of a declarative query.

The visual query language is declarative, not imperative. That is, the dataflow in Fig. 2 is not implemented eagerly, running one filter after the other, as proposed in the Pipes and Filters architecture pattern [30]. Its execution does not take all the sensors in the system and then applies the three filters sequentially. Instead, the language takes the declarative query and creates an efficient routine to populate the list of sensors from the dynamic discovery layer in Fig. 1. This layer is implemented with a database management system that, for instance, indexes the sensors by authorized user. The resulting sensor set could be cached when no device has been changed in the network.

Before creating the room warming service, other two queries are built following the same approach (left-hand side of Fig. 3): Meetings to know the meetings in rooms booked for today, and Heaters that returns the set of heaters in one room. Now, the user is ready to build the room warming service using these three queries.

Services and programs are created with visual programming elements, following an approach similar to Scratch and Blockly. Fig. 3 shows a sample service that starts running at 7:00 am, using an event block. Then, the Meetings query is called to get the meetings booked for today, and a loop iterates through them. A new

thread is created for each room, which waits until 5 minutes before the meeting. Then, all the heaters in that room are set to 21 degrees Celsius, using the Heaters query. The following loop waits 10 minutes and, if the is no one in the room, turns off the heaters and terminates.

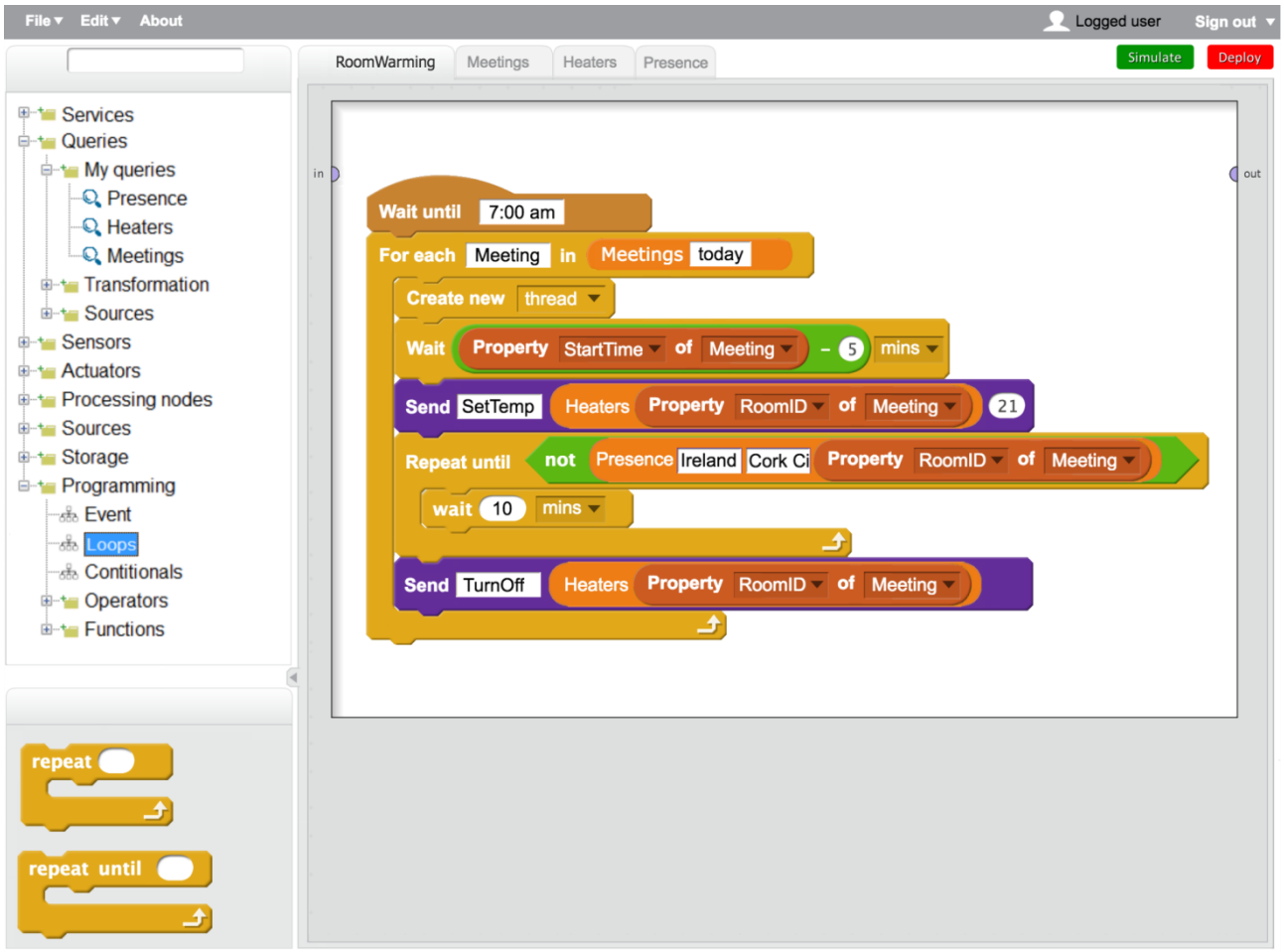

Fig. 3. Creation of an example application.

For each query and service developed, the user may simulate its execution (top right of Figures 2 and 3). Simulation asks the user about runtime events to facilitate testing. If the simulation is correct, the user will then click on the Deploy button. The framework will implement the user-defined microservices (the three queries and the room warming service) in a specific technology (e.g., language and application framework) depending on the existing configuration of the computation node they will be deployed on. It could also use any of the orchestration and choreography tools existing for microservice architectures [23]. Then, the framework transparently selects the best computation node to deploy that service depending on different dynamic criteria such as network topology and traffic, the devices used by the service, and the computation capacity available in the nodes. In addition, non-device microservices are deployed in the cloud, guaranteeing high availability, scalability and accessibility. When necessary, software containers could be used to provide self-deployment of microservices.

If presence sensors or heaters in the building are replaced, the service will continue working, because it is programmed against the Presence, Heaters and Meetings declarative queries. The requirement is that the description of new presence sensors and heaters indicate the room they are located and their purpose. Once the room warming service is created, anyone authorized could execute it using any device (e.g., a smartphone or tablet) connected to the Internet.

4. Related Work

Artem Katasonov identified the eventual necessity of empowering nonprogrammers to easily program the devices connected to smart environments [12]. According to Katasonov, such a system must provide high-level programming abstractions, on-the-fly deployment, flexibility to add and remove devices from the environment, and a mechanism to restrict user access to the existing services. He proposes the use of Ontology Driven Software Engineering (ODSE) to support the composition of components to create applications supported by the graphical Smart Modeler tool [31]. He also identifies the utilization of Scratch as an important direction of future work [12].

Some authors detected the similarities between large-scale distributed applications and IoT environments: they require big scalability and adaptability capabilities to adapt to dynamically changing environments [32]. Since the microservice architecture has been successfully used with distributed systems, it was applied to the design of the DIMMER Smart City IoT platform [32]. DIMMER uses semantic information to provide the dynamic discovery of devices, while microservices support its decentralized management.

Node-RED is a tool for wiring together hardware devices, APIs, and online services connected to the same network [33]. It provides a visual browser-based programming system that generates JavaScript code to be deployed as part of a Node.js server. Unlike our proposed system, NodeRED follows a local programming approach rather than distributed macroprogramming [34]. Although data sources are connected graphically, the user must write JavaScript code when specific processing is necessary. Therefore, it does not provide visual programming for general-purpose programming (Figure 3), and hence the user must be able to program in JavaScript. The query system described in this article has been partially inspired in NodeRED and RapidMiner [35].

Other visual programming languages have been defined to program IoT devices. NETLabTK is a visual drag and drop web interface to connect sensors, actuators, media, and networks associated with smart objects in IoT environments [36]. Ardublock is a graphical programming language for Arduino integrated in Eclipse, while allowing the developer to write Java code [37]. Modkit is another visual language to program different microcontrollers including Arduino, littleBits, Particle Photon, MSP340, Tiva C launch pad, and Wiring S [38]. S4A customizes Scratch to program Arduino devices [39]. miniBloq is an open source graphical programming environment for Multiplo, Arduino, physical computing devices and robots, which allows writing C/C++ and Python text code [40]. All of these languages are focused on programming devices rather than distributed IoT applications that use different devices and services.

There exist different research works aimed at tackling the heterogeneity of IoT applications. One approach is using Model-Driven Development (MDD), as Srijan [10]. Srijan separates different concerns such as domain, platform, deployment and architecture. Different roles in the IoT development are also identified: domain experts, software designers, application developers, device developers and network managers. Each one uses a domain-specific language to specify their concerns, and the final application is generated by combining all the specifications. Other IoT systems based on MDD are PervML [41], DiaSuite [42], ATaG [43] and Pantagruel [44].

Macroprogramming is another approach to deal with the heterogeneity of IoT development. Opposite to expressing the behavior of local nodes, macroprograms describe the behavior of a distributed system. Kairos is provided as a Python extension that hides from the programmer the details of distributed-code generation and instantiation, remote data access and management, and inter-node program flow coordination [8]. Parts of the Python program are compiled to native code and installed in the nodes. A runtime library implements runtime calls between nodes, managing access to each node state. MacroLab is a macroprogramming framework that provides a vector programming abstraction similar to Matlab for

Cyber-Physical Systems (CPSs) [45]. It implements a deploy-specific code decomposition to distribute the operations in the macroprogram across the network.

Olympus is a high-level programming model for pervasive computing environments [46]. It specifies with high-level descriptions the changing resources available in the runtime environment, such as services, applications, devices, devices, locations and users. The framework resolves the high-level descriptions into actual entities based on constraints, ontological descriptions, existing resources, space-level policies and the current context. Then, a system called Gaia divides the application into components that are migrated across the devices in the environment [47].

There are different IoT reference architectures aimed at handling the common requirements of IoT applications, and offer them as a superset of reusable functionalities [48]. The Industrial Internet Reference Architecture (IIRA) is a standard-based open architecture for Industrial Internet Systems [49]. The IIRA has industry applicability to drive interoperability, map applicable technologies, and guide technology and standard development. The description and representation of the architecture are generic and at a high level of abstraction. The IIRA distills and abstracts common characteristics, features and patterns from common use cases, prominently those that have been defined in the Industrial Internet Consortium (IIC).

IoT-A proposes an architectural reference model together with the definition of an initial set of key building blocks [50]. Using an experimental paradigm, IoT-A combines top-down reasoning about architectural principles and design guidelines with simulation and prototyping in exploring the technical consequences of architectural design choices. This architectural model is now being maintained by the IoT Forum.

Fog Computing is a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing Data Centers [51]. It uses collaborative end-user clients to carry out a substantial amount of storage (rather than stored primarily in cloud data centers), communication (rather than routed over the internet backbone), control, configuration, measurement and management (rather than controlled primarily by network gateways). This architecture provides low latency and location awareness, wide-spread geographical distribution, a very large number of nodes, predominant role of wireless access and heterogeneity. Therefore, fog computing has been identified as an appropriate architecture for IoT services and applications [51]. The OpenFog Reference Architecture is aimed at developing an open architecture fog computing environment [52].

5. Conclusions

Although IoT programming is a difficult task, end-users will demand programming their own IoT services and applications. In this position paper, we describe the architecture of an IoT framework to facilitate this task, describing its key elements. We propose visual general-purpose programming to specify the domain logic, and dataflow abstractions to provide data processing and the declarative identification of devices. Semantic self-descriptive information should be provided for each smart object to support the implementation of a global discovery service. A microservice architecture for the proposed IoT framework assists in providing adaptability to physical changes in IoT networks, the usage of different technologies and languages, failure recovery, and the utilization orchestration and choreography tools. A cloud-based environment supporting those microservices eases network traffic optimization, scalability, accessibility and availability. Software containers could be used to transparently deploy microservices in the IoT network.

We plan to implement the propose IoT framework in the following years. Its development will require a multidisciplinary team with knowledge in software development, network administration, embedded devices, programming language design, cloud computing, compiler construction and distributed

middleware.

Acknowledgment

This work has been funded by the European Union, through the European Regional Development Funds (ERDF); and the Principality of Asturias, through its Science, Technology and Innovation Plan (grant GRUPIN14-100). We have also received funds from the Banco Santander through its support to the Campus of International Excellence.

References

[1] Xia, F., Yang, L. T., Wang, L., & Vinel, A. (2012). Internet of things. *International Journal of Communication Systems*, *25(9)*, pp. 1101–1102.

[2] Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, *54(15)*, 2787–2805.

[3] Xu, L. D., He, W., & Li, S. (2014). Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, *14(2)*, 2233–2243.

[4] Gartner newsroom. (2017). Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. Retrieved from the website: http://www.gartner. com/newsroom/id/3598917

[5] Sarkar, C., Nambi, S. N. A. U., Prasad, R. V., & Biswas, A. R. (2014). A scalable distributed architecture towards unifying IoT applications. *Internet of Things*, *WF-IoT*, 508–513.

[6] Madden, S., Franklin, M. J., Hellerstein, J. M., & Hong, W. (2002). Tag: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, *36*, 131–146.

[7] Madden, S., Franklin, M. J., Hellerstein, J. M., & Hong, W. (2003). The design of an acquisitional query processor for sensor networks. *Proceedings of the International Conference on Management of Data*. 491–502.

[8] Gummadi, R., Gnawali, O., & Govindan, R. (2005). Macro-programming wireless sensor networks using Kairos. *Proceedings of the International Conference on Distributed Computing in Sensor Systems*, DCOSS' 05, 126–140.

[9] Alcaraz, C., Najera, P., Lopez, J., & Roman, R. (2010). Wireless sensor networks and the Internet of Things: Do we need a complete integration? *Proceedings of the 1st International Workshop on the Security of the Internet of Things* (SecIoT'10). IEEE, Tokyo (Japan).

[10] Patel, P., & Cassou, D. (2015). Enabling high-level application development for the internet of things. *Journal of Systems and Software*, *103*, 62-84.

[11] Cassou, C., Bertran, B., Loriant, N., & Consel, C. (2009). A generative programming approach to developing pervasive computing systems. *Proceedings of the Eighth International Conference on Generative Programming and Component Engineering*, 137–146.

[12] Katasonov, A. (2010). Enabling non-programmers to develop smart environment applications. *Symposium on Computers and Communications*, 1059–1064.

[13] Thoma, M., Meyer, S., Sperner, K., Meissner, S., & Braun, T. (2012). On IoT services: Survey, classification and enterprise integration. *IEEE International Conference on Green Computing and Communications*, 257–260.

[14] Guinard, D., Spiess, P., Trifa, V., Karnouskos, S., & Savio, D. (2010). Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions on Services Computing*, *3*, 223–235.

[15] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications*

of the ACM, *52(11)*, 60–67.

[16] Wilson, A., & Moffatt, D. (2010). Evaluating scratch to introduce younger schoolchildren to programming. *Proceedings of the 22nd Annual Workshop of the Psychology of Programming Interest Group*, 1–12.

[17] Ray, P. P. (2017). A survey on visual programming languages in internet of things. *Scientific Programming*, 1–6.

[18] Fowler, M., & Lewis, J. (2014). Microservices. Retrieved from the website: http://www.martinfowler. Com / articles /microservices.html

[19] Internet of Things World Forum (2014). Building the Internet of Things. Retrieved from the website: http://cdn.iotwf.com/resources/72/IoT_Reference_Model_04_June_2014.pdf

[20] Thönes, J. (2015). Microservices. *IEEE Software*, *32(1)*, 116–117.

[21] Georgakopoulos, D., Jayaraman, P. P., Zhang, M., & Ranjan, R. (2015). Discovery driven service oriented IoT architecture. *Proceedings of the IEEE Conference on Collaboration and Internet Computing*.

[22] Serrano, M., Quoc, H. N. M., Phuoc, D. L., Hauswirth, M., Soldatos, J., Kefalakis, N., Jayaraman, P. P., & Zaslavsky, A. B. (2015). Defining the stack for service delivery models and interoperability in the Internet of Things: A practical case with OpenIoT-VDK. *IEEE Journal on Selected Areas in Communications*, *33(4)*, 676–689.

[23] Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. 1st Edition, O'Reilly Media.

[24] Pötter, H. B., & Sztajnberg, A. (2016). Adapting heterogeneous devices into an IoT context-aware infrastructure. *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 64–74.

[25] Wang, W., De, S., Cassar, G., & Moessner, K. (2013). Knowledge representation in the Internet of Things: Semantic modelling and its applications. *Automatika - Journal for Control, Measurement, Electronics, Computing and Communications, 54(4)*, 388–400.

[26] Deshmukh, P. P., Pund, B. G., & Pund, M. A. (2012). Cloud computing assure high accessibility, scalability and processing power with windows azure. *IJCA Proceedings on International Conference on Benchmarks in Engineering Science and Technology*, 11–14.

[27] Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*.

[28] Cockcroft, A. (2014). State of the art in microservices. *DokerCon Europe* (keynote talk), DockerCon'14, pp. 1–78.

[29] Lieberman, H., Paterno, F., & Wulf, V. (2006). *End User Development*. Springer.

[30] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern-oriented software architecture. Volume 1: A System of Patterns, Wiley Publishing.

[31] Katasonov, A., & Palviainen, M. (2010). Towards ontology-driven development of applications for smart environments. *Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, Mannheim, Germany, 696–701.

[32] Krylovskiy, A., Jahn, M., & Patti, E. (2015). Designing a smart city internet of things platform with microservice architecture. *Proceedings of the 2015 3rd International Conference on Future Internet of Things and Cloud*, FICLOUD '15, IEEE Computer Society, Washington, DC, USA, 25–30.

[33] Node-RED (2017). A visual tool for wiring the Internet of Things, Retrieved from the website: https: //nodered.org

[34] Blackstock, M., & Lea, R. (2014). Toward a distributed data flow platform for the web of things (distributed node-red). *Proceedings of the 5th International Workshop on Web of Things*. (pp. 34–39).

[35] RapidMiner. (2017). Data science behind every decision. Retrieved from the website:

https://rapidminer.com
[36] NETLabTK. (2017). NETLabTK: tools for tangible design. Retrieved from the website: http://www.netlabtoolkit.org
[37] Ardublock (2017). Ardublock: A graphical programming language for Arduino. Retrieved from the website: http://blog.ardublock.com
[38] Modkit. (2017). Modkit: Programming your world. Retrieved from the website: http://www.modkit.com
[39] S4A. (2017). S4A: A scratch modification for simple programming of Arduino. Retrieved from the website: http://s4a.cat
[40] miniBloq. (2017). An open source programming environment for Multiplo and Arduino. Retrieved from the website: http://blog.minibloq.org
[41] Serral, E., Valderas, P., & Pelechano, V. (2010). Towards the model driven development of context-aware pervasive systems. *Pervasive and Mobile Computing*, *6(2)*, 254–280.
[42] Cassou, D., Bruneau, J., Consel, C., & Balland, E. (2012). Toward a tool-based development methodology for pervasive computing applications. *IEEE Transactions on Software Engineering*, *38(6),* 1445–1463.
[43] Pathak, A., & Prasanna, V. K. (2011). High-level application development for sensor networks: Data-driven approach. *Theoretical Aspects of Distributed Computing in Sensor Networks, Monographs in Theoretical Computer Science*. An EATCS Series, Springer Berlin Heidelberg, 865–891.
[44] Drey, Z., Mercadal, J., & Consel, C. (2009). A taxonomy-driven approach to visually prototyping pervasive computing applications. *Proceedings of the Domain-Specific Languages*. (pp.78-99). IFIP TC 2 Working Conference, DSL.
[45] Hnat, T. W., Sookoor, T. I., Hooimeijer, P., Weimer, W., & Whitehouse, K. (2008). Macrolab: A vector-based macroprogramming framework for cyberphysical systems, *SenSys*, 225–238.
[46] Ranganathan, A., Chetan, S., Al-Muhtadi, J., Campbell, R. H., & Mickunas, M. D. (2005). Olympus: A high-level programming model for pervasive computing environments. *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, PERCOM '05, IEEE Computer Society, Washington, DC, USA, 7–16.
[47] Román, M., Hess, C. K., Cerqueira, R., Ranganathan, A., Campbell, R. H., & Nahrstedt, K. (2002). Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 74–83.
[48] Weyrich, M., & Ebert, C. (2016). Reference architectures for the Internet of Things. *IEEE Software*, *33(1)*, 112–116.
[49] Industrial Internet Consortium (2015). Industrial internet reference architecture. Retrieved from https://www.iiconsortium.org/IIRA-1-7-ajs.pdf
[50] IoT Forum. (2017). Internet of thinks architecture. Introduction to the architectural reference model for the internet of things. Retrieved from http://iotforum.org/wp-content/uploads/2014/09/120613-IoT-A-ARM-Book-Introduction-v7.pdf
[51] Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*.
[52] OpenFog Consortium. (2017). OpenFog reference architecture for fog computing. Retrieved from https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf

Francisco Ortin is an associate professor of the Computer Science Department at the University of Oviedo, Spain. He is the head of the Computational Reflection research group (http://www.reflection.uniovi.es). He received his BSc in computer science in 1994, and his

MSc in computer engineering in 1996. In 2002 he was awarded his PhD entitled *A Flexible Programming Computational System developed over a Non-Restrictive Reflective Abstract Machine*. He has been the principal investigator of different research projects funded by Microsoft Research and the Spanish Department of Science and Innovation. His main research interests include dynamic languages, type systems, aspect-oriented programming, computational reflection, and runtime adaptable applications.

Donna O'Shea is a lecturer in the Computer Science Department of the Cork Institute of Technology, Ireland. She received her BSc in software development & computer networking (2002), a master of engineering by research (2004) and a degree in philosophy (PhD) (2007). She is a member of the Computing research group called Ríomh (Cloud and Distributed Systems Group). Donna's research expertise lies in the area of network and service management with a specific focus on the design, analysis and optimization of wired and wireless communication systems, networks and services. During her PhD she made a significant contribution in the area of service provisioning for Beyond 3rd Generation (B3G) networks.