RapIoT Toolkit: Rapid Prototyping of Collaborative Internet of Things Applications

Simone Mora, Francesco Gianni and Monica Divitini
Norwegian University of Science and Technology
Department of Computer and Information Science
Trondheim, Norway
Email: {simone.mora, francesco.gianni, monica.divitini}@idi.ntnu.no

***Abstract*—The Internet of Things holds huge promises to enhance collaboration in multiple application domains. By bringing Internet connectivity to everyday objects and environments it promotes ubiquitous access to information and integration with third-party systems. Further, connected "things" can be used as physical interfaces to enable users to cooperate leveraging multiple devices via parallel and distributed actions. Yet creating prototypes of IoT systems is a complex task for non-experts because it requires dealing with multi-layered hardware and software infrastructures. We introduce RapIoT, a software toolkit that facilitates prototyping IoT systems providing an integrated set of developer tools. Our solution abstracts low-level details and communication protocols allowing developers to focus on the application logic, facilitating rapid prototyping. RapIoT supports the development of collaborative applications by enabling the definition of high-level data types primitives. RapIoT primitives act as a loosely-coupled interface between generic IoT devices and applications; simplifying the development of systems that make use of an ecology of devices distributed to multiple users and environments. We illustrate the potential of our toolkit by presenting the development process of a IoT system for crowd-sourcing of air quality data. We conclude discussing strength and limitations of our platform highlighting further possible uses for collaborative applications.**

Keywords—Internet of Things, IoT, Ubiquitous Computing, Development, Toolkit.

I. INTRODUCTION

The Internet of Things (IoT), holds huge promises to enhance computer-supported collaboration in several applications domains. By enabling seamless interconnection of people, computers, everyday objects and environments it promotes collaboration off the screen, into our everyday routines. By increasing the amount and quality of information captured by connected objects it might ultimately improve collaboration among people using those objects [1].

Research works have shown how IoT systems can leverage connected objects in collaborative applications; for example, to support patient/physician dialogue in chronic disease treatments [2], to foster social communication among friends and relatives [3], to enhance collaboration in crisis management [4] and to support citizens' participation in public administrations [5].

Yet, since the term Internet of Things was coined in 1999 by technologist Kevin Ashton [6], research has mainly focused on developing machine-centric infrastructures to enable connected things to exchange information over the Internet.

Few works [1], [7] have investigated how IoT can enable collaboration and how HCI theory could drive the development of IoT collaborative systems. Likewise, only few works have investigated collaborative IoT application authoring [8] and how to involve non-experts in design activities [9], [10].

We summarise the characteristics of IoT systems that can support the development of collaborative applications in four areas.

- *Ubiquitous access to information* - IoT's focus on connecting everyday objects using short-range wireless net- works multiplies the number of point of access for information that could be used to support collaboration
- *Integration with third-party systems* - IoT make use of web standards and cloud computing as base technologies [11], enabling integration with established information systems and knowledge bases
- *Physical user interfaces* - IoT can leverage physical and embodied interaction approaches to interact with the "Things". Using physical affordances to interact with computer systems has been proved successfully in supporting collaboration [12, p. 97]
- *Interactions spread among multiple things* - The user experience with IoT is usually distributed on an ecology of devices, providing more opportunities for collaboration via distributed users' actions performed on multiple interfaces.

Notably, while the first two characteristics focus on the *internet* and low-level technology aspects of the IoT, the latter focus on the *thing* aspects; in terms of behaviors and user interfaces.

Prototyping IoT systems is challenging because it requires dealing with a heterogeneous mix of hardware and software components arranged in a multi-layer architecture.

A popular design pattern consists in three layers:

- an *embedded layer* implemented as a physical object augmented with sensors, actuators and short-range wireless connectivity to provide sensing and user interface capabilities

978-1-5090-2300-4/16 $31.00 © 2016 IEEE
DOI 10.1109/CTS.2016.81

- a *gateway layer*, implemented as a device such as smartphone or WiFi router, provide connectivity to the embedded layer enabling ubiquitous access to information
- a *server layer* implemented as a cloud service enables for data storage and integration with third-party services.

As an example, popular wearable fitness tracker products feature a pedometer sensor with a simple user interface to show the number of steps counted or calories burned (embedded layer), a cloud service for aggregating data from multiple users (server layer); and a smartphone app acting both connecting the device to the server layer and as an extended user interface to compare data with other users (gateway layer) (Figure 1). This architectural pattern could be used to implement applications that support collaboration at multiple layers, e.g. by means of both personal or shared multiple devices; which are granted ubiquitous access to information via an infrastructure of multiple gateways.

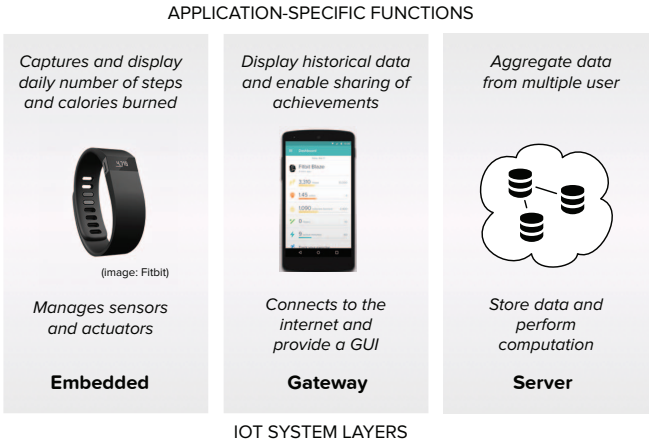

Figure 1. System architecture for a wearable activity tracker, example of an IoT system

Implementing such architecture in working prototypes has for long time required large efforts together with a multidisciplinary team.

Our research aims at supporting rapid prototyping and enabling non-experts in building IoT systems. On one end we aim at lowering the thresholds of skills required to build prototypes; on the other end, we point at raising the ceiling providing extended tools and hacking opportunities to build complex ecosystems.

Although there are a number of tools available to support IoT development, those tools often (i) do not offer an integrated support to multiple architectural layers, (ii) require pre-exiting knowledge in hardware development or embedded programming, (iii) are often bounded to a specific hardware and vendor-lock technologies. This results in a steep learning curve for the tools and large time for integration; obstructing the ability and rapidity to explore design choices by iterating implementing functioning prototypes.

In this paper we present *RapIoT*: an integrated set of tools to support rapid prototyping of IoT applications.

RapIoT does not explicitly support a specific application domain, acting as an enabling technology for the development of collaborative applications by non-experts such as makers, designers and students. In this perspective, RapIoT enables the definition, implementation and manipulation of high-level *data type primitives*. RapIoT primitives allow to abstract low-level implementation details and provide a loosely-coupled interface between different architectural layers. Data types primitive facilitate the development of collaborative applications in two ways.

First they act as a loosely coupled interface between devices and applications, allowing devices to serve different applications without need for reprogramming the embedded layer.

Second, they allow for centralising the application logic in the server layer, offering a platform as a service, thus simplifying the development of systems that make use of an ecology of devices distributed to multiple users/environments.

In the following sections an analysis of existing IoT frameworks and toolkits is provided, the RapIoT approach is then described in detail addressing the technical implementation and flexibility in relation to different application domains. We discuss strengths and weakness of our approach and we conclude the paper highlighting future works.

II. Related Works

Several works have provided tools to facilitate the development of IoT systems by non-experts. Besides relying on standard protocols and APIs that allow mutual integration, each tool often focuses on supporting a specific architectural layer. The knowledge required to use each tool also vary according with the level of abstraction they provide and complexity of the applications that can be achieved. In the remaining of this section we survey development toolkits that can be used for IoT prototyping, considering the barriers that hinder their adoption by non-experts.

A. Development toolkits

In this section we review tools that can be used to support the development of the embedded, gateway and server layer of an IoT infrastructure.

1) Embedded layer: Embedded devices often requires to be programmed with low-level procedural languages which are usually oriented towards production rather than prototyping. On the other side, designers and software developers are usually familiar with high-level, object-oriented programming languages (for example web scripting). For these reasons development tools often provide high-level programming abstractions in the form of proprietary simple textual or visual language or as APIs.

Arduino is a popular prototyping platform which includes both a microcontroller-based board to which sensor and actuators can be wired to; and a software library created to simplify writing code without limiting flexibility [13]. The Arduino library hides developers from learning microcontroller-specific instructions or electronic knowledge. Modkit [14]

extends the Arduino platform providing a block-based visual programming language based on the Scratch project [15], further expanding Arduino target users to non-professional developers such as kids and artists. Focusing on developing interfaces based on simple input/output feedbacks, Bloctopus [16] provides a plat- form based on modules with sensors/actuators couplings and a hybrid visual and textual programming language. Developers can model the behavior of the system taking advantage of both simple visual abstractions and powerful textual commands.

2) Gateway layer: Several research works focused on the gateway layers of the IoT infrastructure. Developing gateways to provide internet connectivity to resource-constrained embedded devices is particularly limiting for non-experts, because it requires pre-existing knowledge of low-level technologies like transport protocols and wireless networks.

McGrath et al. [17] simplify the development and deployment of internet gateways for Bluetooth Low Energy (BLE) devices by abstracting the complexity of dealing with multiple languages and networking aspects. Rather than invoking BLE commands to each local device, their platform provides a proxy to access multiple devices via a centralised API. Yet this approach still requires pre-existing knowledge about the BLE protocol. Also the development of firmware for the embedded layer, to provide custom abstractions or primitives to the programmer is not specifically addressed.

Zhu et al. [18] addresses the development of a gateway for ZigBee wireless devices. Their system is based on three layers: perception, transmission, and application. IoT devices can be controlled and accessed remotely and the gateway handle conversion between different data protocols. Yet this solution implies that only the parent node is connected to the network, and child nodes are not directly accessible through an unique IP address.

3) Server layer: The server layer is the core element that takes care of managing IoT devices connected via multiple gateways as well as interaction with third-party web services such as data providers or social networks.

The framework *PatRICIA* [19] leverages a programming model and a cloud-based execution environment for reducing complexity and supporting scalable development of IoT applications. The solution however focuses on providing sensor management in a cloud environment and storing data received from connected devices; neglecting interaction with other third-party solutions. They also neglect the management of connected devices through an API, and rather focus on reading and combining data from different sources. Each device is directly connected to the cloud through the MQTT protocol[1] excluding mobile and low-powered IoT devices.

Similarly, the framework developed by Khodadadi et al [20] focuses on connecting data sources by managing querying and filtering of data, and facilitating sharing with third-party platforms. Their work take into account data-gathering from multiple sources, both from sensor networks, and from other web applications (blogs, social media, databases). Users are provided with an API for configuring data sources and to trigger actions within stand-alone applications. Kovatsch et al. [21] describes a similar higher-level architecture. They address the need of an API for connected devices for pushing and retrieving data. The proposed solution, which builds on the CoAP protocol[2], enables devices to publish data to third- party servers, but doesn't support bi-directional exchange of events in real-time.

B. *Non-experts as IoT developers*

RapIoT builds on top of Arduino strengths and extend a similar approach to the IoT world. Developers interested in building applications are offered with a set of primitives that are tailored and specific for the affordances of the IoT hardware in use, but at the same time they share a common semantic structure and are used in the same way when coding the application logic. Another point in common is the abstraction of vendor-specific programming mechanisms: like the Arduino user, which is not required to know type and producer of the microcontroller, RapIoT users are not required to know any hardware- or software-related detail of the IoT devices. The user only need to be aware of the set of primitives defined and available to be used for development.

III. RAPIOT FUNDAMENTALS

A. *Design goals*

RapIoT aims at providing a holistic support to the development of IoT systems. The following design goals constitute the foundation of our platform.

Support both novice and expert developers – Provide basic, simple to use functionalities without hindering expert users in building complex systems

Decoupling infrastructure from application – IoT infrastructure is provided as a service to applications. In this way the infrastructure (IoT devices, gateways and server) can be reused across different applications with no or little changes

Hide hardware complexities – Provide high-level representations of low-level embedded hardware complexities

Hide networking details – Spare developers from implementing connection and data transfer protocols

Generic embedded devices – Enable the development of applications that make use of a wide range of devices no matter of manufacturer

Multiple embedded devices – Enable the development of IoT systems that make use of multiple devices which collaborate as a structured ecology

Mobile devices – Support development of IoT systems with mobile devices, e.g. wearables.

We believe that those design goals can be achieved by empowering data primitives. We provide tools to support development and use of primitives across different layers.

[1] MQTT protocol specifications - http://mqtt.org

[2] CoAp protocol specifications - http://coap.technology

B. Input/Output Primitives

RapIoT supports the development of collaborative applications by enabling the definition, implementation and manipulation of high-level *data type primitives*. A RapIoT *input primitive* is a discrete information sensed by an IoT device; for example a data-point captured by a sensor or a manipulation performed via a user interface. An *output primitive* is an action that can be performed by the IoT device via output features such as actuators or displays, for example a motor spinning or a LED (Light Emitting Diode) blinking (Figure 2).

Primitives act as a loosely coupled interface between embedded devices and one or more application logics. Each primitive encapsulates a data type plus up to two optional parameters as payload. Example of an input primitive is "AirQuality (primitive name), city center (parameter 1), low (parameter 2)" in case of an air quality sensor device or "FrontDoor, knocked" in case of a smart home equipped with an accelerometer device on the front door. Otherwise "Necklace, vibrate" represents an output primitive that issues a vibrate command to a necklace equipped with an haptic motor device.

The role of primitives is twofold. On one side they provide an event-driven approach to programming, on the other side they facilitate collaboration among developers working on different IoT layers by providing simple constructs to be used to describe the data exchanged between embedded devices and applications. Furthermore they allow non-experts to think in terms of high-level abstractions without dealing with hardware complexities e.g. "shake, clockwise rotation, free fall" for physical manipulations recognised by accelerometer data.

The definition and implementation of primitives is performed by programming the firmware of an Arduino-compatible device in order to register the primitives. The primitives are then available to the framework and it is possible to implement low-level hardware details; for example, dealing with accelerometer or GPS sensors as well as motor or display actuators.

Primitives not only support simple input/output operations, they can also encapsulate more complex behavior to support the development of physical interfaces; as illustrated in [10]. An example of HCI primitive introduced in [10] is the *"proximity"* input primitive. The primitive does not encapsulate any sensor-data from the surrounding environment, it is triggered when one or more IoT devices are moved close to one another. It is available to be used for devices that have the on-board hardware to support the functionality (i.e. RFID antennas and tags).

Primitives specific for each device can be implemented by using *RapEmbedded* library running on Arduino boards. Instances of primitives are propagated using *RapGateway* smartphone app and accessible from client applications via a simple API provided by *RapCloud*.

C. Architecture

RapIoT composed by:

- *RapEmbedded*: an Arduino library to support definition and implementation of input and output primitives on embedded hardware devices;
- *RapMobile*: a cross-platform mobile app that acts as internet gateway and allows to discover and configure IoT devices;
- *RapCloud*: a cloud service, API and javascript library that support the development of applications that interact with IoT devices.

In the following section we illustrate how RapIoT can be employed to create a simple IoT application.

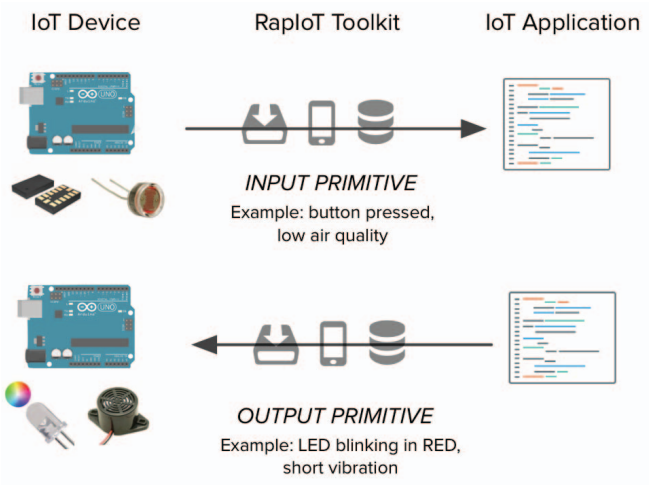

Figure 2. Structure of input and output primitives.

IV. Creating RapIoT Applications

The development of an IoT application using RapIoT is a five-step process. The first three steps entail application development. The last two steps involve application appropriation by end users.

- *Device development* – it involves (i) building a hardware prototype of a IoT device using electronic components on an BLE-enabled, Arduino-compatible board and (ii) use the *RapEmbedded* library to register and implement input/output primitives
- *Application development* – it entails coding application features by using APIs and libraries provided by *RapCloud*, input and output primitives are here employed as programming constructs
- *Application deployment* – it involves uploading an application code on *RapCloud* using a web interface
- *Device appropriation* – it entails wireless discovery of the prototype built in step 1 using the *RapMobile* smartphone app
- *Application appropriation* – it involves selecting an application previously uploaded on RapCloud and running it using the RapMobile app.

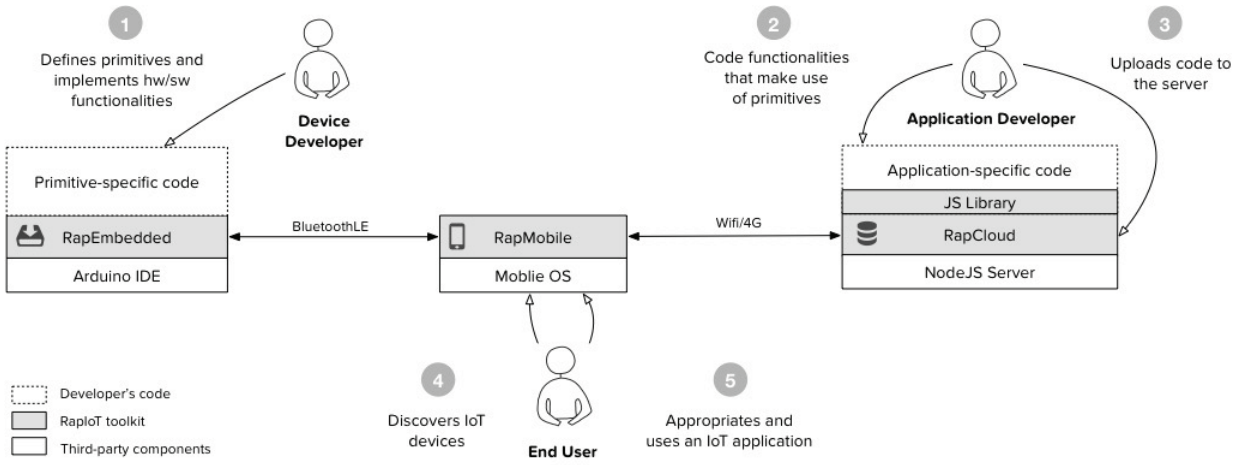

Figure 3. The RapIoT toolkit and development process

The list of steps and their relation with RapIoT components is reported in Figure 3.

To describe the development process of RapIoT applications we introduce as running example the development of *Breath Better Air*, an IoT system to engage citizens in monitoring air quality in their neighbourhoods. This is a collaborative application that relies on individual contributors to generate a community-wide awareness about air quality in the city.

The Breathe Better Air (BBA) system makes use of a IoT device to sense air quality information and provide visual feedbacks to the users (prototype in Figure 4). The device sends data to a server (developed with RapCloud) which computes the average air quality in a city using the data furnished by all BBA users. Eventually, the BBA device provides visual warnings using a green and red LEDs to display whenever the air quality captured by the device is over or above the average value provided by other users.

In the following we describe the BBA application development and deployment process.

A. Device development

Device development involves hardware and firmware development.

Hardware development involves plugging together electronic components using an Arduino-compatible board (Figure 4). To date, RapEmbedded supports a number of development platforms that feature an Arduino-compatible microcontroller and a Bluetooth Low Energy (BLE) chip; such as RFDUINO[3] and Simblee[4] boards. RapEmbedded does not pose limitations on the type of sensors and actuators connected.

Firmware development requires writing Arduino code that interfaces with hardware to generate and consume primitives. The *RapEmbedded* library provides functions to: (i) register device types to enable dynamic application/devices couplings and thus simple application appropriation by end-users, (ii) register primitive definitions according with name of the primitive, type (input or output) and name of (up to two) optional parameters and; (iii) code conditions under which primitives are triggered, in case of input primitives, or consumed, as for output primitives.

According with our example the BBA prototype is assembled using a air quality sensor, a RGB LED (Light Emitting Diode) device and an RFDUINO board. (Figure 4).

After having installed the RapEmbedded library in the Arduino IDE, the *device developer* registers the *BBAdevice* device type and defines one input primitive, *AirQuality*, and one output primitive, LED. The AirQuality primitive models air quality levels, it is triggered by sensor readings continuously provided by the sensor and has one *QLevel* parameter that can assume "Low Quality" or "High Quality" states. The LED output primitive provides the *color* parameter that can assume the "green" and "red" states and control a LED to light up in different colors.

```
RIOTe.regDeviceType("BBAdevice");
RIOTe.regPrimitive(in,"Air", "Quality");
RIOTe.regPrimitive(out,"LED", "Color");
```

Finally the developer codes the loop of conditions under which the input primitives are triggered according with readings from the air quality sensor, and implements how to consume the output primitives by issuing commands to control the LED device to light up in different colors.

```
if(CO2Sensor.read() > threshold)
    RIOTe.trigger("Air", "Low");
else
    RIOTe.trigger("Air", "High");

RIOTe.when("LED", "green", callbG());
RIOTe.when("LED", "red", callbR());
callbG(){digital.write(greenPin,HIGH);}
callbR(){digital.write(redPin,HIGH);}
```

[3] http://rfduino.com

[4] http://simblee.com

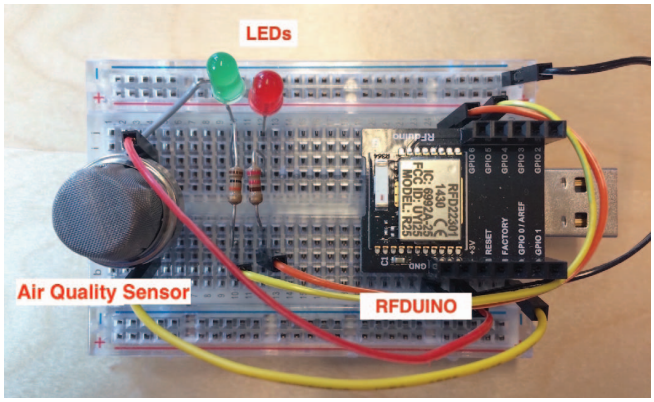

Figure 4. BBA Hardware prototype

After the firmware is developed and deployed, each BBA device is autonomous and ready to establish a connection with *RapCloud* to send and receive primitives (via the *RapMobile* app acting as gateway, as described later).

B. *Application development and deployment*

After primitives are defined and implemented in a (Arduino-compatible) firmware, they are available to application developers from a centralised cloud environment via the RapCloud API. In order to facilitate writing applications, we also developed a javascript library acting as a wrapper for the functionality provided by the RapCloud API.

Back to our BBA example, the application developer proceeds coding the application logic. First she registers the application name and the type of device required. Then she proceeds coding the application logic: whenever the *AirQ* primitive is received, its *QLevel* value is stored in a database (DB). The DB is then queried for average air quality values computed from reading provided by multiple BBA devices. If the current QLevel value is lower than the QLevel average an output primitive is issued to turn the *LED* on the BBA device to the green state; otherwise to the red state (current air quality lower than the average):

```
RapIoTApplication bba;
bba = rIoT.regApp("BBA","BBAdevice");
bba.when("Air",DB.add(bba.Air.Quality));
if(bba.Air.Qiality > DBStore.Average)
   bba.trigger(LED.green);
else
   bba.trigger(LED.red);
```

As a final step the application developer proceeds uploading the source code to the RapIoT cloud server using a dedicated web form. The BBA application is now available to end users.

C. *Device and Application appropriation*

End users are provided with *RapMobile* app, compatible with Android and iOS devices. RapMobile mainly acts as a gateway layer between IoT devices (implemented with RamEmbedded) and the *RapCloud* service; yet it also allows to select and activate applications previously registered with RapIoT.

In order to run the BBA application, the user performs four steps. First the user installs the RapMobile app on her smartphone. Second, she selects the BBA application among the ones available. Third, she discovers and selects the BBA device she wants to associate to the BBA application among the list of Bluetooth devices available nearby. Fourth, she starts the application. Whenever the application is running the phone can be set in standby mode but should remain within a 10 meters reach from the BBA device to ensure reliable data transfer. The GUI supporting appropriation and execution of BBA is shown in Figure 5.

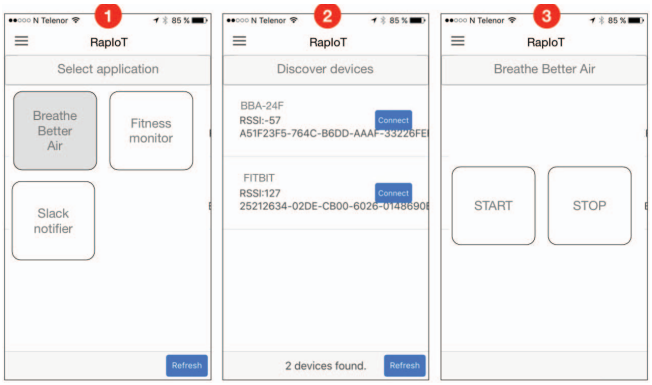

Figure 5. RapMobile Application

V. IMPLEMENTATION

RapIoT is built on top of MQTT and CoAP protocols. Primitives are coded in JSON-formatted messages that contain a unique identifier of a device, currently implemented as MAC address, followed by the identifier of a primitive and the two optional parameters.

Primitives are exchanged between IoT devices and applications on a event-driven basis. The event protocol is very lightweight and designed for low-resource embedded devices, since the information required to route primitives from each device to applications is offloaded by the gateway and server layers. This design choice spares hardware and application developers from implementing event routing, since each hardware module can be unequivocally controlled by an application connected to the API; no matter where the application or the hardware are deployed. Application developers only need to handle input primitives received from the hardware modules and send output primitives to those devices, without the need to know how the modules implement the actual recognition and actuation of primitives.

Our platform employs a broadcast-based architecture in which all embedded devices interact with a common (wireless) communication channel where messages are broadcasted over the MQTT protocol.

This architecture enables the reuse of deployed devices for different applications without changing the firmware. Furthermore, hardware modules can be discovered, attached or removed to the platform while clients are still running. Special system-wide events inform connected clients of the availability of new devices in real time.

The current implementation has several limitations. The web interface for uploading application code on RapCloud is still under development, yet it is possible to launch applications manually from a command line interface. Likewise, RapMobile does not yet fully support selection and execution of applications (steps 1 and 3 in Figure 5); requiring developers to hard-code devices' MAC addresses.

VI. Discussion

In this section we analyse how RapIoT could drive the development of collaborative applications and we discuss its strengths and limitations.

A. Support for collaboration

Our approach to IoT system development embeds mechanisms that facilitate the authoring of collaborative applications. Primitives demonstrated to be a flexible construct that allow to break down interaction routines and data flows into simpler blocks that can be combined when writing the application logic. The RapIoT toolkit presents three fundamental features that help developing collaborative systems:

- *Support for multiple devices* – RapIoT supports applications that make use of several devices connected to the same gateway (*RapMobile* app). This allow multiple users to interact with several devices placed in the same environment, which are then ruled by a centralised application logic running on the *RapCloud* server. Collaborative applications are then a concrete possibility: users can cooperate interacting with different devices for a common goal;
- *HCI primitives for physical interaction* – some of the primitives rely on composite actions and events, which involve more than one physical device. It is possible to design and implement applications that support time coordination, sequential actions, proximity and other forms of cooperative practices that characterize coordinated ecologies of devices;
- *Distributed gateways and devices* – applications developed with RapIoT can use several gateways physically located in different places, each of which can control a group of devices. This open to several possible scenarios: (i) groups of users can move from site to site where different groups of IoT devices are located and perform collaborative tasks that involve IoT devices on the site, i.e. a collaborative treasure hunt game, (ii) users can carry one or more IoT devices connected to their smartphone and perform some tasks or collecting data in the environment, remotely cooperating with other users that are following the same workflow but on a different site, (iii) users can move from gateway to gateway performing a subset of tasks involving different IoT devices, remotely collaborating with other users that follow the same process in other sites, with different IoT devices.

B. Limitations

The RapIoT architecture does not comprehend any coded application logic embedded into IoT devices (embedded layer). Since the primitives have to follow a complete round trip from the embedded layer to the application layer, network latency can be a significant factor affecting performance and application responsiveness. Network quality and availability is crucial for the entire period when the application is in use.

This limitation can be particularly amplified when the application layer deals with batches of primitives in rapid sequence. In these cases, most of the execution time is spent waiting for the network, which can hinder the user experience.

Another possible limitation is connected to the concept of primitive: for some applications the behavior to encapsulate in a primitive can be too complex to be exposed with a simple interface like the one provided by input/output primitives. This restriction could be partially mitigated splitting the logic into two or more primitives, with the drawback of delegating more work to the network.

VII. Conclusions

In this paper we presented the RapIoT toolkit for rapid prototyping of IoT applications. The development process of a RapIoT application has been demonstrated by describing how the provided tools have been applied to the development of a system for crowdsourcing air quality data.

RapIoT leverages the concept of data primitives as a communication block and interface between generic devices and application layers. Further, we highlighted how RapIoT primitives can support the development of collaborative applications via multiple embedded devices, physical interfaces and distributed gateways.

RapIoT takes advantage and builds on top of the most recent technological evolutions in the field like the Arduino platform, cloud computing, BLE radios and mobile applications; reducing complexity and entry barriers for non-experts.

Future works will be oriented towards testing and refinement of the tools composing the system, as well as development of more complex applications and collaboration-specific primitives.

References

[1] O. Eris, J. Drury, and D. Ercolini, "A collaboration-focused taxonomy of the Internet of Things," *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 29–34.

[2] A. J. Jara, M. A. Zamora, and A. F. G. Skarmeta, "An internet of things–based personal device for diabetes therapy management in ambient assisted living (aal)," *Personal and Ubiquitous Computing*, vol. 15, no. 4, pp. 431–440, 2011.

[3] M. Brereton, A. Soro, K. Vaisutis, and P. Roe, "The messaging kettle: Prototyping connection over a distance between adult children and older parents," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 713–716.

[4] L. Yang, S. H. Yang, and L. Plotnick, "How the internet of things technology enhances emergency response operations," *Technological Forecasting and Social Change*, vol. 80, no. 9, pp. 1854–1867, Nov. 2013.

[5] N. Taylor, U. Hurley, and P. Connolly, "Making community: The wider role of makerspaces in public life," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: ACM, 2016, pp. 1415–1425.

[6] K. Ashton, "That 'internet of things' thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[7] T. L. Koreshoff, T. Robertson, and T. W. Leong, "Internet of things: A review of literature and products," in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, ser. OzCHI'13. NewYork, NY, USA: ACM, 2013, pp. 335–344.

[8] M. Blackstock and R. Lea, "Iot mashups with the wotkit," in *2012 3rd International Conference on the Internet of Things (IOT)*, Oct 2012, pp. 159–166.

[9] D. De Roeck, K. Slegers, J. Criel, M. Godon, L. Claeys, K. Kilpi, and A. Jacobs, "I would diyse for it!: A manifesto for do-it-yourself internet-of-things creation," in *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, ser. NordiCHI '12. New York, NY, USA: ACM, 2012, pp. 170–179.

[10] S.Mora, J. Asheim, A.Kjøllesdal, and M.Divitini, "Tiles Cards: a Card-based Design Game for Smart Objects Ecosystems," in *Proceedings of the First International Workshop on Smart Ecosystems cReation by Visual dEsign co-located with the International Working Conference on Advanced Visual Interfaces (AVI 2016)*, vol. 1602. Bari, Italy: CEUR-WS, Jun. 2016, pp. 19–24.

[11] J. Gubbi, R. Buyya, S. Marusic, and M.Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[12] O. Shaer and E. Hornecker, "Tangible User Interfaces: Past, Present, and Future Directions," *Foundations and Trends in Human–Computer Interaction*, vol. 3, no. 1-2, pp. 1–137, Jan. 2009.

[13] D. Mellis, M. Banzi, D. Cuartielles, and T. Igoe, "Arduino: An open electronic prototyping platform," in *Proceedings of CHI Extended Abstracts*. ACM, 2007.

[14] A. Millner and E. Baafi, "Modkit: Blending and extending approachable platforms for creating computer programs and interactive objects," in *Proceedings of the 10th International Conference on Interaction Design and Children*, ser. IDC '11. New York, NY, USA: ACM, 2011, pp. 250–253.

[15] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch Programming Language and Environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, pp. 16–15, Nov. 2010.

[16] J. Sadler, K. Durfee, L. Shluzas, and P. Blikstein, "Bloctopus: A Novice Modular Sensor System for Playful Prototyping," in *TEI '15: Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction*. ACM, Jan. 2015.

[17] W. McGrath, M. Etemadi, S. Roy, and B. Hartmann, "Fabryq: using phones as gateways to prototype internet of things applications using web scripting," in *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 2015, pp. 164–173.

[18] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "Iot gateway: Bridgingwireless sensor networks into internet of things," in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, Dec 2010, pp. 347–352.

[19] S. Nastic, S. Sehic, M. Vo¨gler, H.-L. Truong, and S. Dustdar, "PatRICIA–A Novel Programming Model for IoT Applications on Cloud Platforms," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*. IEEE, 2013, pp. 53–60.

[20] F. Khodadadi, R. N. Calheiros, and R. Buyya, "A data-centric framework for development and deployment of internet of things applications in clouds," in *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, April 2015, pp. 1–6.

[21] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*