# Towards Computational Notebooks for IoT Development

**Fulvio Corno**
Politecnico di Torino
Turin, Italy
fulvio.corno@polito.it

**Luigi De Russis**
Politecnico di Torino
Turin, Italy
luigi.derussis@polito.it

**Juan Pablo Sáenz**
Politecnico di Torino
Turin, Italy
juan.saenz@polito.it

## ABSTRACT

Internet of Things systems are complex to develop. They are required to exhibit various features and run across several environments. Software developers have to deal with this heterogeneity both when configuring the development and execution environments and when writing the code. Meanwhile, computational notebooks have been gaining prominence due to their capability to consolidate text, executable code, and visualizations in a single document. Although they are mainly used in the field of data science, the characteristics of such notebooks could make them suitable to support the development of IoT systems as well. This work proposes an IoT-tailored literate computing approach in the form of a computational notebook. We present a use case of a typical IoT system involving several interconnected components and describe the implementation of a computational notebook as a tool to support its development. Finally, we point out the opportunities and limitations of this approach.



*CHI'19 Extended Abstracts, May 4–9, 2019, Glasgow, Scotland UK*



ACM ISBN 978-1-4503-5971-9/19/05.
https://doi.org/10.1145/3290607.3312963

**KEYWORDS**

Internet of Things; Computational notebook; Documentation; Software development

## INTRODUCTION AND BACKGROUND

Internet of Things (IoT) systems are characterized by a set of common architectural elements, namely: devices, gateways, cloud services, and applications [8]. *Devices* are hardware to collect sensor data (sensing devices) or perform actions (acting devices). *Gateways* collect, preprocess, and forward the data coming from the devices to the cloud, as well as the requests coming from the cloud to the acting devices. The *cloud* manages the devices, acquires and stores the data, and provides real-time and/or offline data analytics. *Applications* vary from web-based dashboards to domain-specific web and mobile applications [9].

However, the implementation of these IoT systems is complex [2], as it requires an unusually broad spectrum of development technologies and skills [9] and involves several programming languages along with their corresponding development and execution environments (which, in turn, depend on the concerned architectural element, its expected usage, and the available computing resources). Against this backdrop, it would be desirable for IoT programmers to rely on interactive documentation that enables them to edit and share textual explanations, as well as executable code, dependencies, and terminal commands, so that the implementation process facilitates, both when writing the code and when configuring the development and execution environments.

Literate programming originates in 1984 from a paper by Donald Knuth. It suggests software developers that *"instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do"* [5]. In line with this proposal, literate computing tools such as computational notebooks have emerged. They are designed to support the construction and sharing of computational narratives by enabling data analysts to arrange code, visualizations, and text in a computational narrative [4, 7].

These literate computing tools are based on cells, each of which contains rich text or code that can be executed to compute results or generate visualizations. These cells are linearly arranged but can be reorganized, reshuffled, and executed in any order. Moreover, programmers can pick and choose which code cells they would like to edit and run [3]. Notwithstanding, notebooks also have limitations: (i) saving application states is difficult, limiting the ability to develop applications from within a notebook, (ii) real-time collaboration is, at best, limited to text editing, and (iii) the behavior of a notebook cannot be reprogrammed or extended from within, limiting its expressive power [3].

Taking into account the IoT software development scenario and the literate computing tools landscape, we aim at proposing an IoT-tailored literate computing approach to support students and novices in the development process of several, interconnected components of an IoT system. This

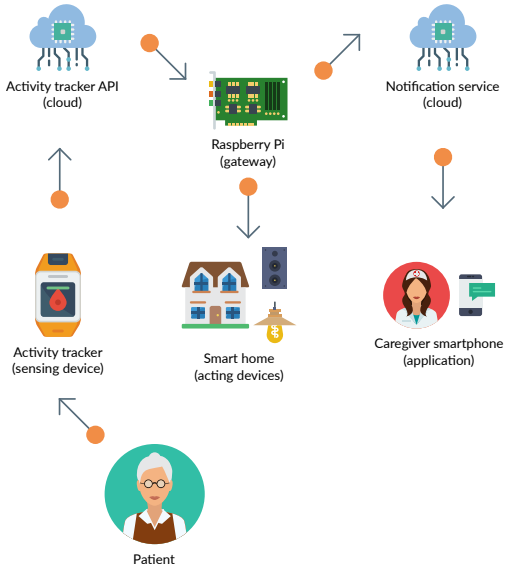


**Figure 1: Emergency Quest IoT system architecture**

goal includes the implementation of an *IoT notebook* in line with the requirements of IoT systems and the evaluation of the usefulness of such an approach. The implementation of heterogeneous IoT components is intended to be captured into a single document where executable code is surrounded by textual explanations and real-time visualizations. Textual explanations in the form of documentation, indeed, demonstrated to be particularly important for IoT systems developers [10]. In this paper, we aim at answering, through a use case, to what extent literate computing tools, given their characteristics, represent a feasible approach to support the IoT systems development, mainly with prototyping and experimenting purposes.

## USE CASE

To assess the feasibility of our proposal, we present a use case consisting of an IoT system that involves the four aforementioned architectural elements. This use case was chosen from the final projects of a university course where undergraduate students worked in groups developing IoT systems [1]. The project is called Emergency Quest and aims at improving the quality of life for people living with mild dementia or Alzheimer while granting them greater autonomy and allowing their caregivers to be aware of their situation. Through a wearable device, the system could monitor the stress levels, location, and activities performed by the patient. If the system detects, via the accelerometers of the bracelet, that the patient becomes agitated, it will react trying to calm her down through the deployment of a relaxing setup that includes music and lighting. Additionally, if the patient abandons a certain area, the system would react and warn her caregivers by delivering notifications on their smartphones. Technically speaking, as illustrated in Fig. 1, this system uses a Fitbit Flex 2 Activity Tracker (sensing device) to monitor the heart rate, location, and activities performed by the patient. Later, this information is gathered by a Raspberry Pi 3 Model B+ (gateway) that determines if the patient becomes agitated and connects with the Philips Hue Lighting System (acting device) and the Sonos One Wireless Speaker (acting device) to deploy the relaxing setup, as well as with the Amazon Simple Notification Service (cloud) to generate and deliver a push notification through a mobile application in the caregivers smartphone (application).

The implementation of this system implies the use of various languages (namely, Java, Python, and SQL queries), the deployment of a software component that keeps executing on the single-board computer, the installation of several dependencies needed for the development of the features (i.e., the AWS, Fitbit, and Philips Hue SDKs), and the interaction with third-party APIs.

Upon these characteristics, we explored the computational notebook landscape to identify the most suitable tools currently available to satisfy them. Concretely, we decided to use Project Jupyter by taking in consideration that: (i) it is currently one of the most widely used platforms [7]; (ii) it is open-source and relies on open standards; (iii) it can be deployed locally; and (iv) it supports several programming languages. These characteristics make Jupyter notebooks suitable for being extended

**Additional features that an IoT notebook must enable**

- *Multiple programming languages* in the same notebook.
- The code in the documents must be executable in *external devices*, such as a single-board computer connected to the USB port.
- Deal with the *always-on nature* of IoT systems. It means that some software snippets should remain on execution, contrary to the sequential manner in which current computational notebooks execute them.
- Support the specification and installation of diverse *dependencies*. Additionally, the code snippets concerning the installation of these dependencies must be labeled as mandatory and be executed at the beginning of the document.
- Support the *visualization* of data coming from the sensing devices or external services and platforms.

**Sidebar 1: IoT notebook features**

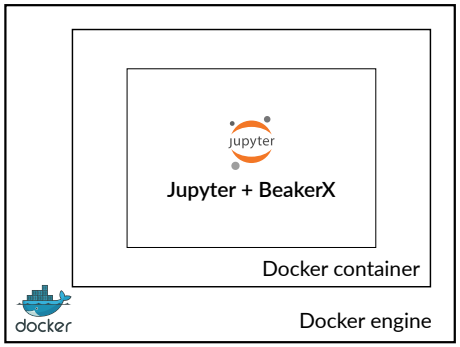


**Figure 2: Docker deployment architecture**

or customized. This analysis leads to the development of an IoT notebook, that besides the features of the current computational notebooks, must enable additional features (Sidebar. 1). Hereafter, a preliminary version of such an IoT notebook, whose UI is depicted in Fig. 3, is described in the context of the depicted use case.

## THE "EMERGENCY QUEST" IOT NOTEBOOK

The architecture of the prototypical system is based on executing the notebook on top of the Docker engine (Fig 2). This container wraps up software and its dependencies into a standardized unit for software development that includes everything it needs to run: code, runtime, system tools, and libraries. In our scenario, Docker helps to avoid environment inconsistencies that could take place when running the code snippets of the notebook in different execution environments.

Once defined the deployment environment, the identified requirements were satisfied in this manner:

**Multiple languages:** we chose the Jupyter interactive computing environment along with BeakerX, a collection of kernels and extensions that provide polyglot programming support: multiple languages in the same notebook [6]. Besides the traditional languages supported by Jupyter, BeakerX adds support for Groovy, Scala, Clojure, Kotlin, Java, and SQL.

**Execution on external devices:** The Docker container engine is suitable to be deployed on a Raspberry Pi. Therefore, our strategy for this initial evaluation of the IoT Notebook approach consists of taking advantage of the Docker features to seamlessly execute portions of the notebook across various devices. However, it requires to install and deploy Decker in the Raspberry Pi.

**Always-on nature:** since the Jupyter notebook provide full support for system shells, the code snippets that had to keep running were embedded into scripts and run as terminal commands.

**Dependencies management** the dependencies installation was easily achieved thanks to the Jupyter support for system shells. For this initial prototype, the most practical way to define the precedence and compulsory nature of some code snippets was through the textual cells, by indicating in the narrative the need to run the code snippets in a given order.

**Data visualization:** the data gathered from third-party services was parsed and visualized using Python scripts.

The prototypical system was able to successfully implement the Emergency Quest use case. Specifically, the integration between services and remote platforms were achieved in this way:

- The integration with the Amazon Simple Notification Service was achieved through the use of the AWS Command Line Interface by running system shell instructions from our notebook.

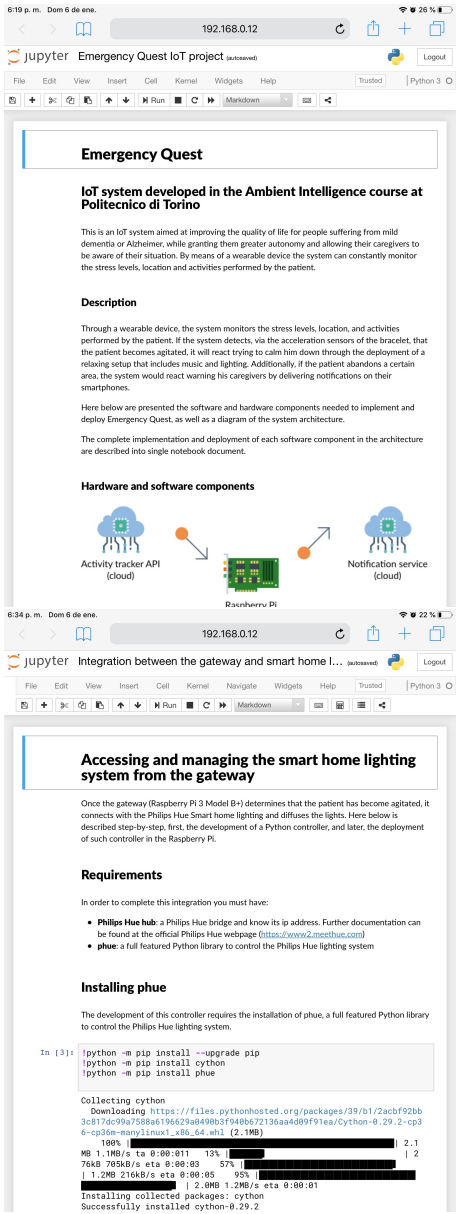


**Figure 3: Screenshots of the IoT notebook**

- The integration with the Fitbit API was possible due to the support that BeakerX provides for JVM and several code snippets implementing the OAuth authentication mechanism were developed. Finally, the data gathered from the API was displayed in the notebook.
- The integration with the Philips Hue hub was achieved through various Python code snippets. All of these code snippets are surrounded by text cells with the description and the explanation of the code.

## DISCUSSION AND FUTURE WORK

Current computational notebooks along with the software stack described in the previous section enable to partially satisfy the requirements identified for an IoT notebook. However, special attention should be paid on how to execute the code snippets on external devices (such as single-board computers) that may be connected wirelessly or physically to the developer's computer. While installing the notebook inside a container and executing it directly on the Raspberry Pi board (as in the prototypical IoT notebook) is a feasible alternative, a smoother technical implementation should be provided so that code snippets can be automatically and remotely executed across several devices, being them Docker-able or not, as it is common in the IoT scenario. This is part of the ongoing work.

Moreover, although the precedence, order, and dependencies among the snippets can be expressed through explanations on the text cells, it would be desirable to have elements in the graphical interface that enable users to set the compulsory execution order of the snippets and their dependencies.

Finally, as future work we will consider:

(1) A more in-depth assessment of the benefits and limitations of a computational narrative in the context of IoT software development and prototyping, with the involvement of multiple IoT novice developers.
(2) The exploration of various alternatives in the implementation of IoT notebooks. For instance, the advantages that reactive notebooks might bring, particularly to track dependencies between cells and automatically determine when to re-run them if something change.
(3) The identification of the scenarios and conditions under which literate computing can be effectively used for prototyping a complex IoT system, e.g., how to handle multi-file projects.

## CONCLUSION

The use of computational notebooks is mainly tied to the data science field. However, given the purpose of these computational notebooks and the characteristics of IoT systems, in this paper, we proposed a literate computing approach to support their implementation and deployment. To assess the feasibility of this proposal with the current computational notebooks, we developed a use case with an IoT system that consisted of four architectural elements: devices, gateways, cloud, and applications.

Through this use case, we identified the opportunities that computational notebooks provide, as well as the limitations that should be addressed by an IoT-tailored computational notebook. Moreover, we presented a prototypical IoT notebook that implements such a use case, and we depicted opportunities and limitations.

## REFERENCES


[1] Fulvio Corno and Luigi De Russis. 2017. Training Engineers for the Ambient Intelligence Challenge. *IEEE Transactions on Education* 60, 1 (Feb 2017), 40–49. https://doi.org/10.1109/TE.2016.2608785

[2] Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz. 2018. Easing IoT Development for Novice Programmers Through Code Recipes. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '18)*. ACM, New York, NY, USA, 13–16. https://doi.org/10.1145/3183377.3183385

[3] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. 2018. The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 174, 11 pages. https://doi.org/10.1145/3173574.3173748

[4] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter development team. 2016. Jupyter Notebooks - a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, 87–90. https://eprints.soton.ac.uk/403913/

[5] Donald E. Knuth. 1984. Literate Programming. *Comput. J.* 27, 2 (May 1984), 97–111. https://doi.org/10.1093/comjnl/27.2.97

[6] TwoSigma Investments LP. 2018. BeakerX. http://beakerx.com. Online; last accessed December 19th, 2018.

[7] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 32, 12 pages. https://doi.org/10.1145/3173574.3173606

[8] Antero Taivalsaari and Tommi Mikkonen. 2017. A Roadmap to the Programmable World: Software Challenges in the IoT Era. *IEEE Software* 34, 1 (Jan 2017), 72–80. https://doi.org/10.1109/MS.2017.26

[9] Antero Taivalsaari and Tommi Mikkonen. 2018. On the development of IoT systems. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. 13–19. https://doi.org/10.1109/FMEC.2018.8364039

[10] Camilo Vieira, Alejandra J. Magana, Michael L. Falk, and R. Edwin Garcia. 2017. Writing In-Code Comments to Self-Explain in Computational Science and Engineering Education. *ACM Trans. Comput. Educ.* 17, 4, Article 17 (Aug. 2017), 21 pages. https://doi.org/10.1145/3058751