IoT environments introduce specific constraints such as lossy networks, low-powered nodes, and communication capabilities limited to small packet sizes and intermittent connectivity. The original actor model was built on a strong coupling between components and cannot directly be transferred to the IoT. For example, the distributed error handling capabilities were not designed with constraints in mind and require adjustment. Furthermore, security mechanisms are not included and left to the runtime environment. However, IoT devices often carry private or critical data, require reliably and should remain resilient against node tampering. Hence, it is critical to provide encrypted communication as well as an authentication and authorization scheme for nodes.

We adapted CAF to the IoT environment by a new, loosely coupled communication layer as depicted in Figure 1. Part of its design is a transactional message-passing based on the request-response model offered by CoAP. This protocol offers the option of reliable message exchange as well as duplicate message detection. Each message exchange is independent and less vulnerable to connection failures than a coherent data stream. Our runtime environment can provide error propagation and mitigation in cases where messages cannot be delivered after multiple retries.

A major focus in our adaption is a new network stack. The default implementation of CAF focuses on locally distributed hardware with many cores. As such, the network stack is built on TCP/IP in a straightforward manner. In contrast, for the IoT, our transactional network stack targets IEEE 802.15.4 or Bluetooth LE. The IP layer deploys 6LoWPAN to sustain IPv6 compatibility, while UDP is used at the transport layer. With regard to security we rely on DTLS for encryption. Further, we are working on an authentication concept for IoT environments based on ID-based cryptography. Corresponding message exchanges will use the request-response model of CoAP.

As the number of devices connected to the IoT is expected to rise significantly in the coming years, we anticipate a severe demand for professionalizing application development in this domain. Consequently, a programming environment that offers a high level of abstraction on native code is needed to build robust and scalable applications of appropriate robustness and performance. We believe that CAF presents a promising advancement towards 'Programming the IoT'—conjointly with the embedded operating system RIOT.

The main purpose of this work is to allow users to do more with their existing devices and things within their homes or at work. This can be achieved by creating applications customized for individual needs on personal mobile devices. The Puzzle framework allows users to visualize the current status of intelligent objects, such as home appliances; how to operate them—for instance, through a voice command; apply the commands to the object; and more generally, to prototype new Internet of Things (IoT) applications through a user-centred design approach in which users create and develop personalized applications.

In order to allow end users to create their own IoT applications and support their extensibility, the main challenges addressed in Puzzle [1] are twofold: how to create a User Interface (UI) that seamlessly support IoT users to develop applications that meet individual needs, and how to create a flexible platform which supports customisation and interoperability of IoT applications. The mobile authoring environment has been designed taking into account how users can foresee the functions to compose and understand how the flow of their applications progresses, adopting a metaphor that is close to users' real life

A Mobile End-User Development Environment for IoT Applications Exploiting the Puzzle Metaphor

by Josè Danado and Fabio Paternò

‘Puzzle’, a mobile end user development framework, allows the end user to opportunistically create, modify and execute applications for interacting with smart things, phone functions and web services. The framework simplifies development of such applications through an intuitive jigsaw metaphor that allows easy composition on touch-based devices. Users immediately identified the utility of this feature, and found it easy to envisage using the framework in various scenarios in their daily lives.
experiences, thus reducing their learning effort and increasing acceptance. On the other end, the platform needs to seamlessly control execution and communication through a plethora of objects and devices – e.g., light, heating, alarm systems, TVs, mobile devices. Consequently, it is important to be able to support interaction and communication between IoT devices and/or with the environment to exchange data and information ‘sensed’ about the environment, while reacting autonomously to events in the ‘real/physical world’, and influencing it by running processes that trigger actions and create services [2].

Supported interaction techniques were developed in collaboration with end-users, and their feedback was considered both to design and develop the UI and scenarios of use. The approach enabled us to avoid putting technology before the needs of the end user; consequently, allowing end-users to voice their requirements during the design and development of Puzzle, resulting in a user-friendly application able to interact with the physical environment.

Figure 1: Authoring and Execution of Puzzle Applications.

In the HIIS lab at CNR-ISTI, with support from an ERCIM post-doc fellowship, we have created Puzzle – a mobile end user development tool and a web-based platform to create applications including web services, phone functions and smart things. The mobile end-user development tool and related user interface (UI) considered current research in end-user development tools, limitations of mobile devices and focus on a user-centred design approach for IoT [3]. Puzzle is based on the metaphor conveyed by jigsaw pieces to stimulate end users to combine existing functions in ways that make sense to them. The decision to adopt that metaphor was based on its usage on other EUD environments, e.g., Scratch. In contrast to other visual languages, Puzzle adopts a higher level approach not just mimicking a traditional language through a graphical metaphor, but providing jigsaw pieces ready to be combined on the go, thus decreasing the learning curve and motivating users to explore and use it. Furthermore, jigsaw pieces were designed to facilitate users to combine them, and to solve errors and conflicts made during their combination.

At the framework level, end-users tinkering with IoT devices can stimulate creating and adding jigsaws as new building blocks to Puzzle, e.g. adding support to communicate with the lights in the house. New building blocks are created in Javascript; including their logic – i.e., inputs, outputs and functions, according to the Puzzle building block description. Once the building block is developed, it is added to a database storing its description to be combined and used in the authoring tool by end-users; thus enabling its execution management in Puzzle applications. During execution, building blocks follow top-down and left-to-right execution flow, where data outputted from the building block corresponding to a previous jigsaw is the input of a building block associated with the following jigsaw.

Use of IoT devices is triggered, during execution, inside a building block and through the use of web-services. Use of web-services provides a standard access to functionalities, hides platform dependent implementation and, consequently, provides the required interoperability in an IoT environment. Interoperability is also enforced through the Puzzle building block descriptions, which are used to check whether input and output jigsaw data types match and manage the use of unpredicted values during application execution. This is important as the IoT is a heterogeneous environment; thus failure of one device needs to be mitigated by the environment.

Due to the heterogeneity of the environment, standard web-based technologies – e.g., HTML, CSS, Javascript and JSON – are used to support execution of Puzzle applications, wireless networks are used to support communication between IoT devices – e.g., Bluetooth or IEEE 802.15.4, and open hardware is used to foster contributions by a community of users – e.g., Arduino. Open hardware can be used to interface between proprietary devices and Puzzle, allow Puzzle to control IoT devices and be an incentive for a community of users –i.e., tinkering with current hardware - to increase hardware supported, and even create new solutions able to exploit Puzzle connection to existing hardware.

References:

Please contact:
José Danado, Fabio Paternó
CNR-ISTI, Italy.
E-mail: danado@isti.cnr.it, fabio.paterno@isti.cnr.it

Links:
HIIS Lab: http://giove.isti.cnr.it
Framework information at: http://giove.isti.cnr.it/tools/Puzzle/home
Documentation: http://hiis.isti.cnr.it:8080/AuthoringTool/v2/docs/
Video: https://www.youtube.com/watch?feature=player_embedded&v=Asbqv7-QqqM