Towards a Toolkit for the Rapid Creation of Smart Environments

Thomas Kubitza(✉) and Albrecht Schmidt

University of Stuttgart, Stuttgart, Germany
{thomas.kubitza,albrecht.schmidt}@vis.uni-stuttgart.de

Abstract. With the rise of rapid physical prototyping tools such as Arduino it has become very easy for designers, makers and developers to build smart devices, simple installations or other single device solutions. However, as soon as a room, floor or building-wide (prototype-) installation should be build consisting of various types of devices that need to communicate, the effort for building these environments still remains extremely high. A lot of this is due to three factors: programming for different platforms, bridging different communication technologies, and physically connecting devices to network and electricity. In this paper we present a concept that drastically reduces this efforts. Thus, designers and developers can focus more on the implementation of the behaviour of interactive environments. We have implemented this concept as a toolkit for on-site setups that allows to easily mash-up heterogeneous sets of devices using a common scripting language and a web-based IDE. We report from interactive installations in office and museum environments that have been realized based on this platform and we point towards new ways of programming environments.

Keywords: Smart environments · End user programming · Mashups

1 Introduction

Smart and interactive spaces are based on a common principle; different kinds of devices with sensors and actuators attached are statically installed in rooms, levels, whole buildings or are even worn by users. All these heterogeneous devices need to talk to each other or to an entity that constantly combines system state and generates system reactions. Multiple reasons make the setup of such environments a complex task. Firstly, similar functionality has to be implemented on various platforms ranging from microcontrollers to High-End computers. This requires expert knowledge in very specific programming languages and platforms as well as the management of various development environments (IDEs). Secondly, different communications technologies, protocols and formats have to be bridged so that devices can actually exchange data. Thirdly, devices have to be deployed in their target environment, supplied with electricity and wired or wireless communication infrastructure (e.g. WiFi access points). Especially the first two reasons put up high boundary for non-experts in electronics and programming. This limits its usage to small and mostly only professional user groups.

P. Díaz et al. (Eds.): IS-EUD 2015, LNCS 9083, pp. 230–235, 2015.
DOI: 10.1007/978-3-319-18425-8_21

We believe that the right tools can open up the creation of smart environments to a much larger audience in the same way as physical prototyping platforms such as Arduino made the access to microcontrollers much easier and in the same way as Apps made potentially everyone the programmer of his own cell phone (and the phones of millions of others). By empowering groups such as user experience designers, scientist, designers, artists, makers and hobbyists we envision the creation of a large set of truly useful applications evaluated in realistic environments and addressing a broad range of problems.

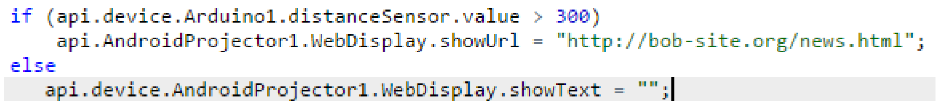
In this paper we present a toolkit that drastically reduces the technical complexity for creating and programming smart environments. Our approach consists of two main pillars: (1) A client software for each type of end device is provided which allows to remotely access and control all its abilities and to abstract from its specific platform. (2) A server software on a central computer node is provided to bridge between various communication-technologies and to provide unified access to all sensors and actuators of configured devices through a web-based JavaScript development environment.

This approach allows to quickly implement or change the behaviour of a system without the need to reprogram or physically access any of the associated or deployed devices. JavaScript, one of the most widespread and growing programming languages, is used to define the system behaviour in a single spot.

A simple scenario is used in following paragraph to illustrate the toolkit usage.

2 Simple Usage Scenario

Bob wants to set up a small WiFi enabled projector in the office floor that should inform about local events. He wants to use a distance sensor connected to a WiFi enabled Arduino that is mounted on the wall to turn on the screen only when somebody is close. He has installed the server part of our toolkit on a spare computer. With the help of the web-based user interface he installs the client firmware on the Arduino device and an App on the Android based WiFi projector. Both devices automatically pop up in the user interface and Bob continuous with adding a "display module" to the projector and a "IR distance sensor" to the Arduino from a list of available modules (see Figure 2a). Now, he switches to the "rules"-view and creates a new behaviour rule consisting of the following JavaScript code:



```
if (api.device.Arduino1.distanceSensor.value > 300)
    api.AndroidProjector1.WebDisplay.showUrl = "http://bob-site.org/news.html";
else
   api.device.AndroidProjector1.WebDisplay.showText = "";
```

Fig. 1. Bobs JavaScript code for controlling a projector using a distance sensor

During typing an auto-completion feature helps Bob to choose the right device-names, module-names and properties (Figure 2b). Bob saves his new rule and approaches the sensor deployed in the floor. When he is in a distance closer than 1.5 meters the news page of his website is displayed; as soon as he leaves the distance a *black screen is shown. Bob decides to tweak the distance value in his rule so that the content is only shown when somebody is closer than roughly 1 meter.*

Two major strengths of our approach are pointed out in this scenario: The central unified access to completely different devices and their easy mash-up. Access to devices is implemented via objects (`api.device.`). This allows using the full power of the JavaScript language and thus the implementation of short and simple rules as well as very complex ones (low threshold, high ceiling).

3 System Concept and Implementation

Conceptually our toolkit strictly follows a master-slave architecture; the client side software allows to treat each client device as a source of sensor events, a sink for actuation commands, or both. It allows abstracting from the underlying operation system and hardware as well as the communication technology and protocol. For each end device platform a specific client firmware has to be implemented. However, this implementation effort is done just once by experts for this specific platform; after that, users of our toolkit just need to install the firmware once and from that point on they can access and control all its abilities from the central server node.

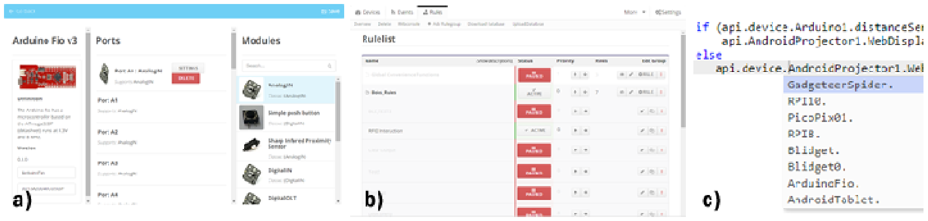
The central server node provides a web-based user interface for the configuration of devices and the creation of behaviour rules. It includes a rule engine that triggers events and runs rules as soon as sensor data is received. Rules may then again trigger actuator commands which are instantly sent to the appropriate devices. In our example a change of the distance sensor value triggered the execution of Bob's rule, which resulted in the actuation of the Wifi projector output if the value was within a certain range.

The system components of the toolkit consist of one computer that runs the server software, an arbitrary number of client devices and an optional external network infrastructure (e.g. WiFi access points). A number of sensor and/or actuators can be attached to a client device (depends on a devices' abilities).

The server software is fully implemented in NodeJS - a framework specialized on the implementation of high-performance multi-protocol applications. This allows the server to be installed on any computer platform (e.g. Windows, Linux) and even on platforms with very limited resources (e.g. Raspberry Pi). The server node can be equipped with various (wireless) communication adapters such as Ethernet, Wifi, Bluetooth / BLE, XBee and RF Link which all work in parallel. The centralized approach allows to completely hide the network layer from the developer. Auto-discovery mechanisms allow a client to find a local server immediately after installing its client software. The web-based interface gives access to all functionalities from the device configuration to rule creation and monitoring. Its web-based nature especially allows to run the development environment on tablets; those can be taken into a smart environment and used to monitor and modify behavior on-site.

A good range of popular client devices is already supported at this stage. The list includes various Arduino devices, .NET Gadgeteer devices [1] and modules, Raspberry Pi, Beaglebone, Intel Edison, various Bluetooth Low Energy devices as well as Android and iPhone smartphones, tablets and projectors. The client software of these devices have the form of firmware, PC software or Apps and expose the full functionality of a device to the central server. Communication protocols and APIs between server and clients are well documented to allow an easy future integration of yet unsupported devices.

4 User Interface



```
if (api.device.Arduino1.distanceSe
    api.AndroidProjector1.WebDispl
else
   api.device.AndroidProjector1.Wel
              GadgeteerSpider.
              RPI10.
              PicoPix01.
              RPI8.
              Blidget.
              Blidget0.
              ArduinoFio.
              AndroidTablet.
```

Fig. 2. a) Device configuration view, b) Rules overview, c) Code auto-completion

The toolkit user interface runs in any browser and is structured in three sections: Devices, Events and Rules. New devices automatically appear in the device overview after the installation of their specific client software. In its initial state a device has no modules configured; this means no sensor or actuators are activated yet. In the device configuration view, modules can be selected from a list of supported sensors and actuator modules (Figure 2a). By double-clicking on a module, it is automatically assigned to a compatible and free port; this way users don't need to care about the right attachment of sensors beforehand, the device configuration steps already give visual hints about the right ports to connect to. After saving a configuration, it is instantly pushed to the device where the selected modules are activated.

In the rules-view an overview of available rules is given. These are structured in groups and consist of a name, a description and an execution priority. Groups allow to tie together rules which are logically associated or only apply to a certain space in the environment (e.g. "Office Floor 1"). Groups and rules can be enabled and disabled individually which allows easy instant switching of behaviour for single spaces or whole environments. Single rules can be edited any time and the changes are applied immediately. Syntax highlighting and auto-completion features help novice users to shimmy along available devices, modules and their individual properties without any previous knowledge; advanced users benefit from the coding speedup and correct referencing of objects. The following section briefly describes two exemplary applications in different environments that where realized using our toolkit.

5 Applications

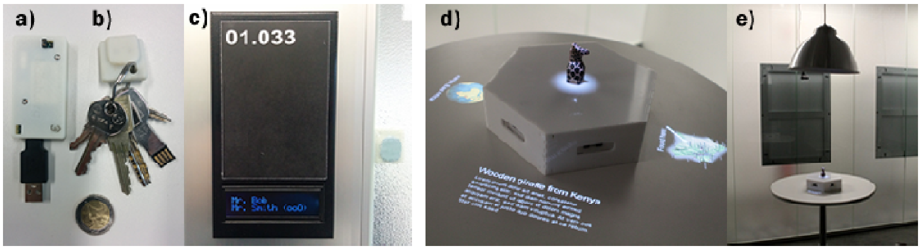


Fig. 3. a) Intel Edison BLE scanner, b) BLE key fob worn by users, c) Electric Imp doorscreen, d) Plinth with distance sensors, e) Projector lamp above plinth

The first setup is created in an **office environment**. It consists of 10 door screens and the same number of BLE scanner devices and BLE key fobs (Figure 3abc). Each key fob is associated to a person and the door-screen in front of this persons' office always indicates whether and where within the building the person is currently located; an "out of office" indication is given when the key fob of a person has not been seen for a while. This behaviour has been realized with a single rule consisting of 45 lines of code. The system is in daily use since multiple months.

The second example setup consists of an **installation for museums**. Three hexagonal plinths are placed within an exhibition room. Each is accompanied by a Pico projector hidden in a lamp above each plinth (Figure 3d-e). The plinth itself is equipped with a RFID reader and six IR distance sensors; this allows recognising exhibition objects on top of the plinth as well as movement (direction and distance) of people around the plinth. The projector permits projecting arbitrary content onto and around the plinth. One use case is the adaption of exhibit related content based on direction and movement of people around. Another use case lets museum visitors put replicas on the plinth in order to receive related digital information on the surrounding surface.

6 Related Work

We focus the discussion of related work on frameworks for rapid protoyping and software for the creation of mash-ups and web-based programming.

Two popular frameworks for the rapid physical prototyping of smart devices are Arduino and .NET Gadgeteer [1]. Latter, provides an ecosystem of modules that can be simply attached without any need for soldering. Both provide a desktop based IDE for programming and deploying firmware on single devices. In contrast to our toolkit these platforms focus on the creation of single standalone devices. However, our toolkit integrates both platforms and its module ecosystems. Other systems focus on the integration of massively widespread smartphones and their integrated sensors as sources for data [2]. Further, peripheral connectors for phones such as IOIO or WebClip [3] allow the attachment of external sensors and actuators.

The idea to use a centralised web hub for mashing up devices in local environments using simple rules has been introduced in [4] and [5]. The system was based on HTTP request, supported very simple sensor to actuator relations and didn't abstract from hardware or communication technology. In [6] a rather simple scripting language has been introduced for smart home setups; by using JavaScript in our toolkit we cover these very simple "If-this-then-that" use cases but also allow for the creation of highly complex rule sets. Another recent trend is the movement of full development environments into the web [7]. The strengths of this approch are for example demonstrated by the mBed IDE; a web-based toolchain for ARM microcontrollers.

7 Conclusion and Future Work

In this paper we have presented a toolkit for the creation and programming of smart and interactive environments that reduces much of the technical complexity and allows users to focus on programming the system behaviour. By choosing JavaScript as scripting language and providing a single web-based IDE we intend to empower a large user group, from novice to expert users, to create and experiment with their own smart environments.

This toolkit is part of an EU project and it is in ongoing development. Two major medium-term milestones are: the implementation of an easy sharing mechanism of smart setups for an online community using "recipes" as well as the experimentation with code that is automatically generated by physically demonstrating actions to smart environments. Further, assistive and visual programming aids (such as Blockly) that can lie on top of the JavaScript layer are currently evaluated. These approaches will be evaluated with cultural heritage professionals from three of our partner museums who will take the role of the smart environment developers.

Acknowledgements. This work is funded by the European Project meSch (http://mesch-project.eu, Grant Agreement No. 600851).

References

1. Hodges, S., Taylor, S., Villar, N., Scott, J.: Prototyping connected devices for the internet of things. IEEE Computer **46**(2), 26–34 (2013)
2. Shirazi, A., Winkler, C., Schmidt, A.: SENSE-SATION: an extensible platform for integration of phones into the Web. IoT (2010)
3. Kubitza, T., Pohl, N., Dingler, T., Schmidt, A.: WebClip: a connector for ubiquitous physical input and output for touch screen devices. Ubicomp, pp. 387–390 (2013)
4. Holloway, S., Stovall, D., Lara-Garduno, J., Julien, C.: Opening pervasive computing to the masses using the SEAP middleware. IEEE Pervasive, pp. 1–5, March 2009
5. Holloway, S., Julien, C.: The case for end-user programming of ubiquitous computing environments. FoSER 2010, p. 167 (2010)
6. García-Herranz, M., Haya, P., Alamán, X.: Towards a Ubiquitous End-User Programming System for Smart Spaces. J. UCS **16**(12), 1633–1649 (2010)
7. Kubitza, T., Pohl, N., Dingler, T., Schneegaß, S.: Innovations in Ubicomp Products Ingredients for a New Wave of Ubicomp Products. IEEE Pervasive Comp., pp. 5–8 (2013)