Apps for Environments: Running Interoperable Apps in Smart Environments with the meSchup IoT Platform

Thomas Kubitza(✉)

University of Stuttgart, Stuttgart, Germany
thomas.kubitza@vis.uni-stuttgart.de

Abstract. With *Apps* a popular concept was introduced allowing end-users to easily extend their devices such as smartphones or computers with specific functionality. Two million Apps have ever since found their way into each of the popular App-stores Google Play and Apple Store. We argue that the App-concept is not only well applicable to single devices but also to complete environments equipped with smart networked things. In the moment when Apps can be easily downloaded and executed in home, office and industry environments a wide new applications space will be opened up. In this work we introduce the concept of *Smart Space Apps* that can be downloaded from a cloud-based App-store into a smart environment where they dynamically utilize the capabilities of available smart things to optimally achieve the purpose they were installed for. We introduce a unified schema for the access of sensors and actuators of heterogeneous devices from within Smart Space Apps and describe the middleware and runtime that implements this approach. We explain how Apps are packaged into an exchangeable format and published within a cloud-based App-store. Multiple application use cases are shown and challenges of this novel approach are discussed.

Keywords: Internet of Things · Smart environments · Middleware · Smart Space Apps

With the increasing number of smart devices, networked sensors and programmable actuators many novel opportunities arise through their smart *composition*. Internet of Things (IoT) technologies and networked wearable devices provide new opportunities for creating distributed tangible user interfaces and intelligent behaviour of networked devices sharing the same physical space. For instance, an activity tracker worn by many users today is mainly collecting activity-data throughout the day, however, when its user is sitting on the couch in front of the TV the embedded accelerometer of the activity tracker *could* be exploited to detect arm gestures that control the TV.

As soon as individual smart things are able to offer some or all of their *capabilities* to the smart environment in which they are located a multitude of new useful applications will arise that optimally make use of this distributed sensors, actuators, input devices and screens to assist users during their onsite activities and tasks. At the same time we believe that the concept of *Apps*, which is a popular approach allowing users to easily extend the functionality of single devices, can be also applied to smart environments.

In this work we report from our ongoing research on realizing this vision. With Apps for Environments we introduce a concept, infrastructure and implementation in which exchangeable Apps that users can download into their smart environments can dynamically utilize available capabilities of networked devices to interact with onsite users and to optimally achieve their purpose.

1 Smart Space Apps Concept

Apps for Environments or *Smart Space Apps* do not run on single devices but instead in environments. For this purpose our approach requires one (stationary) computing node in the local network to run a *middleware* and *runtime* for the execution of smart space Apps. This node can be part of an existing networking infrastructure (e.g. WiFi, Ethernet) or span its own networks using different communication technology adapters (smart hub). A main pillar of our Smart Space Apps concept is a *unified view* on networked computing devices and their *capabilities*: In general, if a device has a processor and a communication interface it can become part of our smart space. Whether certain capabilities of a device should be exposed to the smart environment or not is a decision that inhabitants of a smart space have to decide on inclusion of a new device to the network. Applied to the initial example - the user that controls his TV using his activity trackers arm band, has at some point decided to expose his trackers accelerometer to be used on demand by his domestic environment. In addition he has downloaded and installed a Smart Space App into his smart hubs runtime that promises to utilize devices of the type “activity-tracker” to control devices of the type “TV”. After download and execution the functionality is instantly available. The core functionality of the users’ activity tracker, measuring and reporting activity, is still continued but in addition sensor-data events measured by the accelerometer are shared with the users’ smart environment as soon as the tracker is in its wireless range.

Unified Access Schema
Our unified view on capabilities of smart things breaks these down into *sensors* (devices that are pure data producers), *actuators* (devices that are pure command receivers) and combinations of both. This generic view allows to integrate networked devices of very different kinds in the same way: Capabilities of commercial IoT devices, networked DIY sensors, smart phones, tablets or home appliances can be accessed in the same way. This forms the basis of a *unified access schema* that is used within our runtime to give Apps access to the shared capabilities of an environment. This schema is illustrated in Fig. 1.

This schema uses a (locally) unique user-defined or auto-generated name to reference specific devices, device-wide unique names for modules that represent sensors, actuators, or combinations and names for properties of the referenced module to access sensor data or to trigger actuator commands. As indicated by the dot-notation a runtime could inject this hierarchically structured information into its namespace and execute Smart Space App code written in any programming language that uses this access schema to implement behaviour that interweaves sensor events with system state and actuators. Using the devices indicated in Fig. 1 a minimal Smart Space App could be implemented

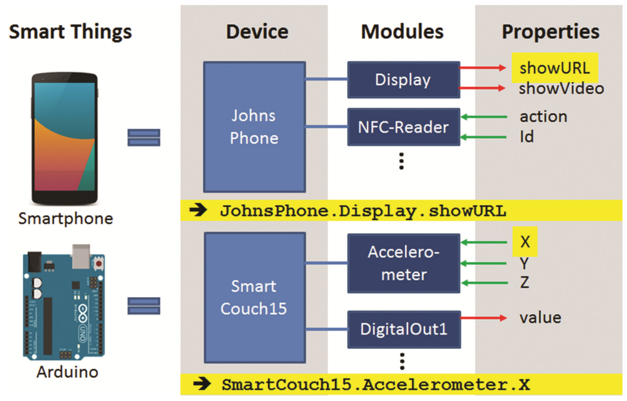


Fig. 1. Generic modularization approach and naming scheme used for referencing smart things and their capabilities

with the following code that shows the weather forecast on the display of “JohnPhone” when the accelerometer of “SmartCouch15” is actuated (e.g. when someone sits down).

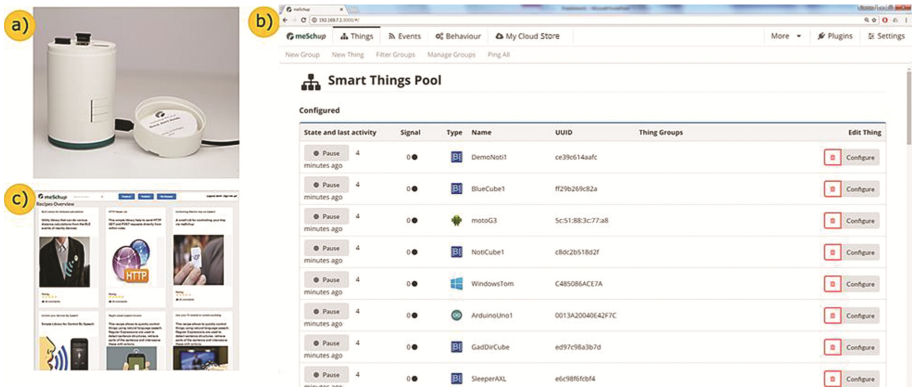
```
if (SmartCouch15.Accelerometer.X > 10) {
     JohnsPhone.Display.showURL ("http://bing.com?q=weather”);
}
```

This very simplistic example illustrates the conceptual level on which logic for Smart Space Apps is implemented. Note that the networking layer is completely abstracted and that devices and capabilities of entirely different platforms and operation system are accessed in the same way. Although very basic, these three lines of code already implement a small Smart Space App that interweaves two specific devices. Complex Apps can consist of thousands of lines of code, interweaving arbitrary numbers of devices, sensors and actuators. In contrast to the example above Smart Space Apps that should run not just in one specific environment, typically do not consist of code that references devices *specifically* (e.g. “JohnsPhone”) but instead reference devices dynamically by their type and capabilities (see section “Runtime” for an example). This allows building and publishing Smart Space Apps that can be downloaded and run in smart environments that consist of very different devices than the ones they were originally build and tested in. For instance a Smart Space App that notifies users when movement sensors were triggered could use flashing the floor-lights in smart space A while in smart space B the same App would show (in addition) a notification message on the displays of currently near tablet or smartphone devices.

In the previous paragraphs we explained the concept behind Smart Space Apps. In the next section we will briefly sketch the implementation of this concept in the form of the meSchup IoT platform.

2 meSchup IoT Platform

The *meSchup IoT platform* [1, 2] was designed and implemented within the four year FP7 EU project *meSch*. meSchup consists of an integrated *middleware, runtime* and web-based *development environment (IDE)* software for onsite hub computers as well as client software, firmware and adapters for a wide range of client devices. The client support at the time of writing incudes Android devices (smartphones, tablets, projectors, TV, Amazon Fire) smart things platforms such as Arduino, Espressif, .NET Gadgeteer and nRF51822 microcontrollers, Raspberry Pi and Intel Edison computers, Windows and Linux machines, smart-plugs, ambi-lights and multiple other smart appliances. For devices that by default are not able to expose their capabilities to local smart environment Apps, client software or firmware is provided. For devices that are not extendable with Apps or firmware such as typical smart X appliances adapters are offered that run in the meSchup middleware and allow direct communication or control of these devices. The meSchup server software is fully implemented in Node.js and can be thus run platform independent on various operation systems. Although meSchup can run on any off-the-shelf machine we provide a set of different meSchHub devices that come optimally preconfigured for various purposes. Figure 2a for instance depicts the meSchup prototyping hub (meSchHub-P) that is based on a Raspberry Pi II computer and comes with its own embedded WiFi Hotspot, battery and meSchup platform preinstalled on SD-card.



Smart Things Pool

Configured

State and last activity	Signal	Type	Name	UUID	Thing Groups	Edit Thing
Pause 4 minutes ago	0●		DemoNoti1	ce39c614aafc		Configure
Pause 4 minutes ago	0●		BlueCube1	ff29b269c82a		Configure
Pause 4 minutes ago	0●		motoG3	5c:51:88:3c:77:a8		Configure
Pause 4 minutes ago	0●		NotiCube1	c8dc2b518d2f		Configure
Pause 4 minutes ago	0●		WindowsTom	C485086ACE7A		Configure
Pause 4 minutes ago	0●		ArduinoUno1	0013A20040E42F7C		Configure
Pause 4 minutes ago	0●		GadDirCube	ed97c98a3b7d		Configure
Pause 4	0●		SleeperAXL	e6c98f6fcbf4		Configure

Fig. 2. (a) meSchHub-P device, (b) meSchup web interface: Overview of configured and recently discovered smart things, (c) Public Smart Space Apps in the cloud-based App store

Middleware

The middleware layer of the meSchup platform abstracts from arbitrary (wireless) communication technologies and protocols and provides unified bidirectional access to remote devices and their capabilities for higher layers of meSchup such as the App-runtime, GUI or IDE. Adapters for various communication technologies are supported such as WiFi, ZigBee, Z-Wave, BLE and LoRa. For each of these communication technologies automatic device-discovery is implemented. Discovered devices are listed in the web-based smart things pool overview where users can configure which of their capabilities they want to provide to their smart environment (Fig. 2). Known devices are automatically discovered and configured every time they return into the (wireless) network. This in particular allows roaming devices such as smartphones to be alternately used in different smart environments. Event-based communication is the default communication model between meSchup clients and the middleware server. This means that client devices do the "hard job" locally such as fast sampling of sensors or complex computation. Only when adequate sensor changes are detected clients will send sensor events to the meSchup middleware. This will trigger Apps to be executed in the meSchup runtime. However, other behaviours such as time-series data can be also configured. Instant remote reconfiguration allows to expose new sensors and actuators anytime without the need to reinstall a client or re-flash firmware of a smart device. This is particularly important to minimize the maintenance for potentially hundreds of devices that are expected in near future smart environments [3]. The event-based communication model saves wireless bandwidth, keeps communication responsive and scales for large numbers of devices.

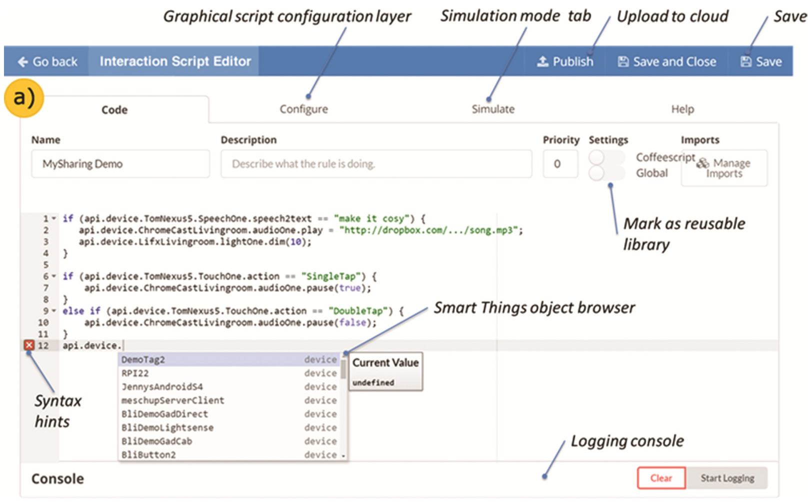
The meSchup middleware is particularly designed to be extendable and strongly inclusive. New communication technologies and protocols can be easily added as middleware modules. By default popular IoT protocols such as MQTT and HTTP/REST are supported by the platform. Virtual devices optimally map the sensor- and actuator-topics/resources of these protocols onto the unified device-module-property access schema.

Web-Based User Interface

meSchup comes with its own web-based GUI for device- and App-management as well as an integrated development environment for the creation, testing and packaging of Smart Space Apps. After starting the meSchup software or powering up a meSchHub the interface is instantly accessible via "http://meschup". The smart things pool view of the UI gives an overview of currently available smart things (Fig. 2b), their connection status, configuration and new devices that have recently been discovered. New sensor/actuator modules can be added or removed in a drag&drop manner and are made instantly available on the remote device. The events view allows monitoring of sensor and actuator events for debugging purposes. The behaviour view provides an overview of currently installed Smart Space Apps and allows to enable/disable, remove or edit Apps (and their underlying interaction scripts). Foreign Apps can be downloaded from a cloud-based Smart Space App store and new Apps can be developed within the integrated IDE and be subsequently published to the store (Fig. 3).

Runtime

The meSchup runtime uses *JavaScript* as the programming language for the implementation of Smart Space Apps. An App can consist of one or many *interaction scripts*. The previously introduced unified access schema is exposed as the object `api.device` within the runtime and provides all Apps unified access to all devices and capabilities of the smart things pool. An exemplary interaction script is shown below. This generic



Name	Description	Priority	Settings	Imports
MySharing Demo	Describe what the rule is doing.	0	Coffeescript Global	Manage Imports

```
if (api.device.TomNexus5.SpeechOne.speech2text == "make it cosy") {
   api.device.ChromeCastLivingroom.audioOne.play = "http://dropbox.com/.../song.mp3";
   api.device.LifxLivingroom.lightOne.dim(10);
}

if (api.device.TomNexus5.TouchOne.action == "SingleTap") {
    api.device.ChromeCastLivingroom.audioOne.pause(true);
}
else if (api.device.TomNexus5.TouchOne.action == "DoubleTap") {
    api.device.ChromeCastLivingroom.audioOne.pause(false);
}
api.device.
```

```
DemoTag2              device
RPI22                 device
JennysAndroidS4       device
meschupServerClient   device
BliDemoGadDirect      device
BliDemoLightsense     device
BliDemoGadCab         device
BliButton2            device
```

Current Value
undefined

Fig. 3. Integrated web based development environment (IDE) for Smart Space Apps

interaction script displays the incoming message on any device that offers a *display-capability* to the local environment.

```
...
// Iterate over all smart things
for (var device in api.device) {
    // Show msg on screen if device offers a display
    if (hasModule(device,"display")) {
        api.device[device]["display"].showHtml ("<b>Msg for John:</b>"+msg);
    }
}
```

Interaction scripts are executed event-based when new sensor events are received by the middleware or timer-based to react on schedules. The developers of App-interaction scripts decide on which events to listen. However, this can be restricted on installation of Apps by not granting access to certain device or module types. The runtime allows executing many Smart Space Apps in parallel. This makes it easy to extend environments continuously with specific functionality required by their inhabitants.

A Smart Space App can optionally provide a graphical configuration user interface that allows End-users to fine-tune its behaviour to their needs and local environment without requiring programming knowledge. These interfaces are web-based and will be typically accessed via users' smartphones or tablets. Developers of Smart Space Apps decide how extensive configuration options are and whether they want to provide a configuration interface at all.

App Store

Smart Space Apps package interaction scripts and optional resources that are for instance required for configuration UIs (HTML, CSS, images, etc.) together with meta-data into exchangeable compressed Smart Space Apps files (.S2A file extension). Meta information includes among others the App-title, description, tags, author and version. Further, information on necessary device types and modules is included. The packaging process is integrated into the web-based IDE. Packaged S2A-Apps can be uploaded into our cloud based App store where they can be easily found by interested users, if the author decides to make them public. The App store requires a previous registration of a user account. This account can be then assigned to one or many meSchHubs that are managed by the same user or organisation. Packaged Apps are then automatically uploaded to this cloud-based App store account but are not made public. Users can either decide to make their Apps publicly available or to keep them private. Users or organisations managing multiple meSchHubs can use the App store to remotely deploy new or updated Apps to one or many of their hubs at once. We currently extend this basic functionality of our App store with additional community functions such as author- and App-ratings, comments, bug-reports and feature requests.

3 Example Applications

meSchup is used as core IoT platform for realizing novel IoT applications in multiple different projects and domains. Three application examples from these projects are briefly described to indicate the broad range of current and future potential application areas for Smart Space Apps.

Smart Interactive Exhibitions
meSchup is actively used in multiple European museums and cultural heritage institutions as the core onsite infrastructure for the realisation of smart interactive exhibitions. Multiple of these exhibitions were packaged up and are available in an App store[1] that was particularly designed for cultural heritage professionals (CHPs). This App store conceptually works in a similar way as the previously described generic App stores but in addition it provides support for the typical workflows of exhibitions designers. In particular it provides easy access to content sources for CHPs such as Europeana[2] and allows to interweave this content conveniently with the smart behaviour of available smart exhibition Apps (these are called *recipes* within mesch.io).

The example application depicted in Fig. 4 for instance is deployed across multiple points of interest (POIs) within a museum exhibition and allows a visitor to perceive personalized multi-media content at each POI based on an object that the visitor has picked at the entrance. This object represents a perspective on the exhibition (e.g. in the context of a First-World-War exhibition narrations from the perspective of a soldier versus the one of a civilian) and the chosen language of the visitor. At each POI a pulsing round area indicates that visitor can place objects on it. This explicit interaction triggers audio, video or other presentations that are tailored to the visitors' language and perspective.

Technically this interactive setup is realized with NFC readers, projectors, screens and earpieces embedded into each POI and NFC tags embedded into the wearable objects for visitors. Apps running in the onsite meSchup platform interweave events from NFC

[1] http://mesch.io.
[2] http://www.europeana.eu/portal/en.

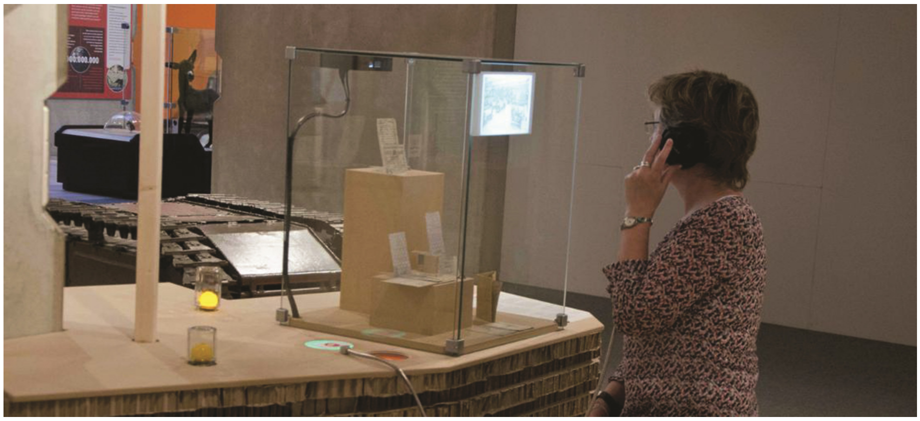


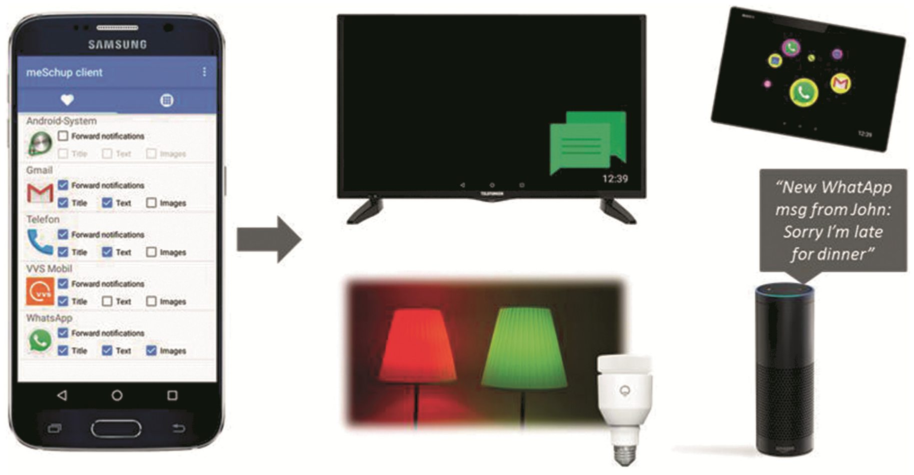
Fig. 4. This point of interest in an interactive exhibition space provides personalized multi-media content using a personal NFC-based token that visitors use across an exhibition

readers with the appropriate content and instantly trigger the display of media at the corresponding points of interest. Associations of content to POIs and NFC objects are not hardcoded but instead use simple semantic tags to derive which content should be shown in which situations. Besides implementing this visible behaviour for visitors the responsible App can also collect anonymised usage data and provide spatial and temporal usage-visualisations for curators.

This example illustrates how meSchup and the Smart Space App concept facilitate the realisation of advanced distributed, customizable interactive installations for novel museum experiences without requiring any low-level programming experience from museum-curators and exhibition designers. Furthermore, installations realized in one physical setup can be easily transferred into another physical setup by just installing the same App on another smart hub. Full interactive exhibition floors can completely switch their purpose and content by simply installing another Smart Space App.

Ubiquitous Notifications

meSchup is also utilized in research projects that explore ubiquitous notifications in the context of smart home and office environments as well as ambient assisted living. In the ubiquitous notifications project [4] meSchup is used to display notifications that arrive at users smartphones (e.g. WhatsApp, Email, Calendar, etc.) instantly in the users environment by dynamically using the capabilities that the current devices in proximity offer. Notifications can thus be overlaid on a TV screen in one situation while they are read aloud by an Amazon Echo device or indicated with ambient lights in another situation. Multiple mobile and stationary deployed sensor sources such as BLE beacons, movement sensors and phone status and orientation are utilized to optimally derive the best devices and output modalities for displaying notifications. Different Smart Space Apps provide an easy way to deploy and test different visualisation concepts and strategies in different physical setups and with different users (Fig. 5).



Android-System
☐ Forward notifications
☐ Title ☐ Text ☐ Images

Gmail
☑ Forward notifications
☑ Title ☑ Text ☐ Images

Telefon
☑ Forward notifications
☑ Title ☑ Text ☐ Images

VVS Mobil
☑ Forward notifications
☑ Title ☐ Text ☐ Images

WhatsApp
☑ Forward notifications
☑ Title ☑ Text ☑ Images

Fig. 5. Ubiquitous notifications: Notifications received on users' smartphones can be instantly forwarded to smart devices in a users proximity, such as TVs and screens, ambient lights, photo-frames and tablets as well as speakers (e.g. Amazon Echo).

In a similar approach the DAAN project[3] utilizes meSchup to realize scenarios for ambient assistive living environments in which contextual information from sensors in the environment are continuously collected and analysed in order to provide ambient proactive suggestions for assistance and behaviour change [5].

Extension of Conventional Things with Smart Interfaces
meSchups instant access to different capabilities of smart things in the same environment allows flexible combinations of sensors, input devices and actuators that physically belong to separate devices. This allows using one or multiple input devices spontaneously to control actuators of other devices. Associations between input and output can be dynamically established using for instance physical tokens such as NFC tags, BLE tags or visual markers. For instance lying down a smartphone on an NFC tag placed in a smart living room could present a dynamically generated GUI for controlling living-room devices. Using a camera and display of a smartphone in combination with markers placed on various devices could overlay these with new or additional interfaces. Figure 6 for instance shows an example in which a small remotely controllable lamp is extended with an Augmented Reality (AR) switch interface that allows turning the lamp on or off.

The concept of such smart interfaces is applicable to many different domains and scenarios: In industrial settings conventional physical instruments (e.g. temperature and pressure gauges) can be augmented with additional data or functionality. For instance overlays of historical graphs (e.g. the temperature curve of the last 24 h) can be shown in addition to the current gauge-value. In medical environments touchless interaction via AR interfaces can help to keep devices sterile. The shown AR interfaces example is

[3] http://daan.dfki.de/.

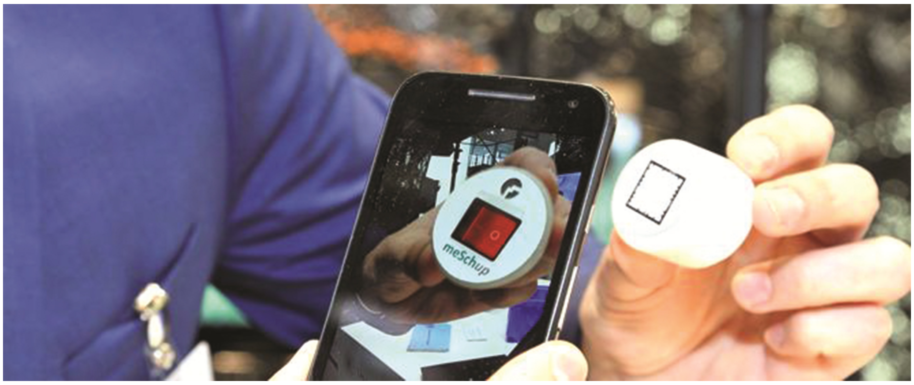


Fig. 6. Augmented Reality Interface extending a smart mobile lamp with a virtual on/off switch. Pressing the virtual switch on the screen instantly turns the lamp on or off.

realised using a Smart Space App that dynamically overlays identified visual markers with graphical interfaces (images). These are chosen depending on the capabilities of the device that is associated with the marker. Overlays are packed with Smart Space Apps and are thus easy to update and to distribute.

4 Related Work

meSchups' underlying concept lies at the heart of traditional Ubicomp research and was inspired by many of its former research projects.

On a device level a set of platforms have laid the way for simplifying the creation of **sensor actuator equipped smart things**. Smart-its [6] have early allowed experimentation with sensors and actuators in academic research. As one of the first commercially available physical prototyping toolkits Phidgets [7, 8] enabled access to electronic components through a pluggable hardware design and easy to use libraries. D.tools [9] allowed designers to iteratively create physical UIs using a state-chart based programming model. These were followed by the .NET Gadgeteer platform [10] with it solderless pluggable module design and powerful Visual Studio IDE as well as the Arduino [11] platform with its simple to setup code editor and huge community support. The "App" concept introduced with Symbian feature-phones and continued by iOS and Android finally also opened up phones to be used as sensor actuator rich platforms that can run user defined code. These systems and platforms facilitated the creation of *standalone* smart devices by simplifying the development of embedded software and the access to sensor and actuator hardware.

meSchup's concept on device level differs from these previous approaches by offering generic ready-to-go firmware/client-software/Apps for all these device platforms instead of supporting developers to build custom firmware for each and every thing. The purpose of this generic firmware is to make the device and its capabilities (sensors and actuators) accessible through its network interface and to be discoverable and fully controllable by the local smart hub and the Smart Space Apps that run on it.

This approach shifts the individual high level application logic into the hubs of an environment and provides unified access to the *capabilitie*s of heterogeneous devices independent of their communication technology, used communication protocol or device platform. The generic firmware already handles all platform specific low level calls, local sensor sampling as well as the secure transport of messages from or to a device. Sensor changes of connected devices simply appear as events within the App runtime of the smart hub and actuator commands are simple function calls within an App. This results in a drastically reduced time and effort for developing distributed logic for heterogeneous IoT devices.

The realization of **distributed multi-device interaction** was addressed by a multitude of systems in particular in the domain of context aware computing and smart spaces. iStuff [12] allowed the prototypic exploration of various novel ubicomp scenarios providing easy means for connecting input and output of distributed devices. Dey et al.'s. framework for prototyping context-aware applications [13] laid the foundations for further projects targeting at enabling end-users. iCap [14] provided a pen based interface that allowed end-users to realise individual context aware applications by graphically composing simple event-condition-action rules. Among others centralized architectures were also proposed in iStuff [12] and SEAP [15]. However, these used rule-based languages that were limited in their expressiveness and complexity. More recently some projects have also picked up JavaScript for executing inter-device behaviour. Fabryq [16] supports mobile scenarios using smartphones as gateways to some BLE devices and hosts the JavaScript based application logic in the cloud. Weave [17] focuses on simplifying the synchronisation of GUIs across multiple device displays.

While most of these projects represent proof-of-concept implementations or toolkits for prototyping that cannot be easily transferred out of the lab, meSchup stands for a highly modular generic IoT platform that provides a wide support for available IoT device platforms and high flexibility through its App based runtime. Its capabilities of providing support from local App-development, to App-packaging, App-Store upload and deployment and execution of the same App on other smart hubs is unique among current IoT systems and research projects.

Multiple cloud based platforms exist that offer end-users simple to use interfaces for the realisation of trivial IoT scenarios. The IFTTT service [18] for instance offers a form based web UI that allows users to compose simple trigger-action rules binding one event to one action. Slightly more advanced interconnections between data sources and data sinks can be created with the flow-based visual programming interface of the Node-Red toolkit [19]. However, for more complex applications the graphical flow based UI expands quickly in space and becomes hard to handle.

In contrast to purely cloud based approaches meSchup by design executes its application logic locally in its runtime and is thus robust against internet connection problems and high latencies. Further, the owner of a smart hub has full control over the data collected through Apps and is not forced to send any data to external parties. meSchup Apps written in JavaScript can realize anything from simplest IFTTT rules using a single sensor and an actuator to complex applications that utilize the full power of the JavaScript language while interacting with hundreds of sensors and actuators. Optionally meSchup Apps can push visual GUIs to devices with displays such as smartphones and tablets. These dynamically generated or static GUIs can be used as configuration layer that makes Apps adjustable to end-users without programming skills. Using pure web-technology for Smart Space Apps is a design decision that allows the huge community of web-developers to instantly start the development of IoT applications.

5 Discussion

The concept of *Apps for Environments* for the first time brings easy extendibility with new functionality to smart multi-device environments in the same way as Apps extend a smartphones functionality. This is achieved by providing a unified interface and an access schema that can map and address arbitrary sensors and actuators of heterogeneous remote devices in a unified way. In a similar way operation systems such as Android or iOS provide unified APIs to control the capabilities of various devices and hardware types using the same App. However, smart environments differ in their exponentially higher potential heterogeneity. All devices in a smart environment as well as their capabilities can be used by multiple Smart Space Apps in parallel. This can in particular result in conflicts where the same actuator resource is manipulated by multiple control sources which can lead to unexpected behaviour. Conflict detection and resolution mechanisms are required to handle such situations. MeSchups' UI offers a specific monitoring view in which such cases are detected and indicated by the runtime and can be resolved by end-users. Our approach allows a user to resolve such conflicts by either deactivating one of the conflicting Apps completely or by specifically withdrawing one of both Apps the right to control the conflicting actuator.

Another challenge in smart environments is the omnipresent disruption of connectivity through wireless connection loss or battery drain. Mobile devices can physically get out of wireless range, be interfered by other signals or suffer high delay through low bandwidth or high traffic. A smart environment needs to be able to detect such situations, handle situations in which devices become unavailable and recover devices and their actuator states as soon as they return. meSchup handles situations in which devices become unavailable pragmatically by dropping all actuator commands that are triggered by Apps as long as the target device is not available. App developers can also implement special event-handlers that are executed when devices become available again, allowing for individual initial configurations for certain device types.

We believe that the most challenging aspects in relation to smart environment and Smart Space Apps are privacy and security. Smart environments offer a multitude of new data sources to collect, analyse and derive personal information about their inhabitants. In contrast to cloud based approaches for controlling smart environments (e.g. IFTTT[4]) the meSchup platform offers data privacy by design because it does not require any communication with external servers. Whether Smart Space Apps can communicate with external sources needs to be fully transparent and controllable by smart space inhabitants to assure the privacy of their data. In meSchup users who install new Smart Space Apps have the opportunity to restrict the access to resource-types of the local

[4] https://ifttt.com/.

smart space as well as to the internet (by default blocked). Currently meSchup offers individual white and blacklists for device types. However, our experience from various deployments indicates that capabilities of smart devices should in the future be annotated with a metric for privacy and security criticality. For instance doorlocks, cameras and microphones would fall into a higher criticality group than for instance switches or lamps. Such groups would then make it easier for users to decide when granting rights on installation.

Mechanisms are further required to assure the functionality, quality and harmlessness of publicly downloadable Smart Space Apps. Installing untrusted Smart Space Apps potentially contains a multiple of privacy and security risks compared to traditional single-device Apps: Cyber-physical systems such as doors, climate control or heating can be potentially locked, stopped or misused. MeSchups' App packaging allows App creators to sign their App with their certificate, ensuring that unmodified Apps can be installed from trusted parties. Further our App store soon allows the rating and commenting by users, thus indicating the satisfaction with a publicly available App.

To prevent theft of sensible sensor data or spoofing of critical control commands within the networks of smart environments end-to-end encryption is necessary on-top of the existing encryptions mechanisms of the individual communication technologies. MeSchups' middleware addresses communication security in three layers. The first layer involves transport layer security such as WPA for WiFis and password based encryption for BLE and ZigBee. On a second layer architectural security means are applied. For instance smart things only accept messages from the same hub after they have been discovered. On a third layer all communication is in addition encrypted with a per-device-key that has been exchanged on initial inclusion of a smart thing to a smart environment. For instance for Android devices a QR-code is scanned to exchange an initial encryption key by-passing unsecure RF communication (out-of-band). Similarly microcontrollers with installed meSchup firmware and USB/Serial/NFC interface can be attached to the hub for a few seconds to exchange a key securely. Encryption on application layer prevents theft of data and guarantees the identity of previously included devices.

meSchups security roadmap further plans the usage of secure elements base on elliptic curve cryptography (ECC) both on future hub-hardware as well as IoT devices equipped with meSchup firmware. This will simplify the secure initial key exchange among these devices.

6 Conclusion

In this paper we describe the concept of Apps for Environments and its implementation as Smart Space Apps based on the meSchup IoT platform. We introduce a unified schema for accessing capabilities of smart things as the foundation for Smart Space Apps and describe its implementation on top of a middleware and runtime. We explain the implementation of Apps, their packaging into an exchangeable format and the distribution via a cloud based Apps store. We introduce multiple example applications spanning across different domains and present challenges of this novel approach.

We believe that the concept of Apps for Environments has an enormous potential to bring innovative applications into current and future smart environments. Our Smart Space App programming approach provides a clean abstraction from low level layers and is fully based on web-technologies such as JavaScript, HTML and CSS. In combination with the integrated development environment and cloud store support we believe that Smart Space Apps are an attractive platform for a broad range of developers. A wide uptake of this concept would lead to a multitude of new useful IoT applications and creative multi-device solutions.

Acknowledgements. This work is funded by the European Project meSch (Grant Agreement No. 600851).

References

1. Kubitza, T., Schmidt, A.: Towards a toolkit for the rapid creation of smart environments. In: Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. (eds.) IS-EUD 2015. LNCS, vol. 9083, pp. 230–235. Springer, Cham (2015). doi:10.1007/978-3-319-18425-8_21
2. Kubitza, T., Schmidt, A.: Rapid interweaving of smart things with the meSchup IoT platform. In: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing Adjunct - UbiComp 2016, pp. 313–316. ACM Press, New York (2016)
3. van der Meulen, R., Rivera, J.: Gartner says a typical family home could contain more than 500 smart devices by 2022. http://www.gartner.com/newsroom/id/2839717
4. Kubitza, T., Voit, A., Weber, D., Schmidt, A.: An IoT infrastructure for ubiquitous notifications in intelligent living environments. In: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing Adjunct - UbiComp 2016, pp. 1536–1541. ACM Press, New York (2016)
5. Wiehr, F., Voit, A., Weber, D., Gehring, S., Witte, C., Kärcher, D., Henze, N., Krüger, A.: Challenges in designing and implementing adaptive ambient notification environments. In: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing Adjunct - UbiComp 2016, pp. 1578–1583. ACM Press, New York (2016)
6. Beigl, M., Gellersen, H.: Smart-its: an embedded platform for smart objects. In: Smart Objects Conference (2003)
7. Greenberg, S., Fitchett, C.: Phidgets: incorporating physical devices into the interface. In: Proceedings of UIST 2001, pp. 209–218. ACM Press (2001)
8. Greenberg, S., Fitchett, C.: Phidgets: easy development of physical interfaces through physical widgets. In: Proceedings of the 14th annual ACM symposium on User interface software and technology - UIST 2001, p. 209. ACM Press, New York (2001)
9. Hartmann, B., Klemmer, S., Bernstein, M.: d.tools: integrated prototyping for physical interaction design. In: IEEE Pervasive Computing (2005)
10. Villar, N., Scott, J., Hodges, S.: Prototyping with microsoft .net gadgeteer. In: Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction - TEI 2011, p. 377. ACM Press, New York (2011)
11. Arduino: Physical prototyping platform. https://www.arduino.cc
12. Ballagas, R., Ringel, M., Stone, M., Borchers, J.: iStuff: a physical user interface toolkit for ubiquitous computing environments. In: Proceedings of the conference on Human factors in computing systems - CHI 2003, p. 537. ACM Press, New York (2003)

13. Dey, A., Abowd, G., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum. Comput. Interact. **16**, 97–166 (2001)
14. Dey, A.K., Sohn, T., Streng, S., Kodama, J.: iCAP: interactive prototyping of context-aware applications. In: Fishkin, K.P., Schiele, B., Nixon, P., Quigley, A. (eds.) Pervasive 2006. LNCS, vol. 3968, pp. 254–271. Springer, Heidelberg (2006). doi:10.1007/11748625_16
15. Holloway, S., Stovall, D., Lara-Garduno, J., Julien, C.: Opening pervasive computing to the masses using the SEAP middleware. In: 2009 IEEE International Conference on Pervasive Computing and Communications, pp. 1–5. IEEE (2009)
16. McGrath, W., Etemadi, M., Roy, S., Hartmann, B.: Fabryq. In: Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS 2015, pp. 164–173. ACM Press, New York (2015)
17. Chi, P.P., Li, Y.: Weave: scripting cross-device wearable interaction. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI 2015, pp. 3923–3932. ACM Press, New York (2015)
18. IFTTT: 'If this then that' cloud service. https://ifttt.com
19. Node-Red: A visual tool for wiring the Internet of Things. http://nodered.org