HAL
open science

Early Lessons from the Development of SPOK, an End-user Development Environment for Smart Homes

Joelle Coutaz, Sybille Caffiau, Alexandre Demeure, James L. Crowley

To cite this version:

Joelle Coutaz, Sybille Caffiau, Alexandre Demeure, James L. Crowley. Early Lessons from the Development of SPOK, an End-user Development Environment for Smart Homes. Ubicomp'2014 , 2014, Seattle, United States. pp.895-902, 10.1145/2638728.2641559 . hal-01492556

HAL Id: hal-01492556

https://hal.science/hal-01492556

Submitted on 20 Mar 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Early Lessons from the Development of SPOK, an End-User Development Environment for Smart Homes

Joëlle Coutaz
University of Grenoble.
Laboratory of Informatics of Grenoble (LIG), France
joelle.coutaz@imag.fr

Sybille Caffiau
University of Grenoble.
Laboratory of Informatics of Grenoble (LIG), France
sybille.caffiau@imag.fr

Alexandre Demeure
University of Grenoble
Laboratory of Informatics of Grenoble (LIG), France
alexandre.demeure@inria.fr

James L. Crowley
University of Grenoble
Laboratory of Informatics of Grenoble (LIG), France
james.crowley@inria.fr

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

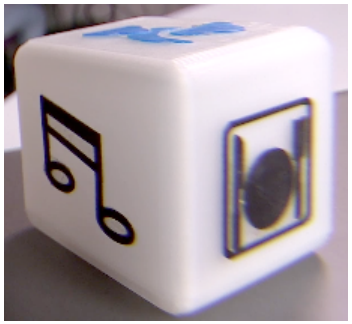
Copyright is held by the owner/author(s).

UbiComp '14, Sep 13-17 2014, Seattle, WA, USA
ACM 978-1-4503-3047-3/14/09.
http://dx.doi.org/10.1145/2638728.2641559

Abstract
This paper presents early lessons from the development of SPOK, an End-User Development Environment for smart homes. SPOK (Simple PrOgramming Kit) uses a pseudo-natural language as an end-user programming language and runs on top of an extension of OSGi/iPOJO to support the dynamic and resilient management of web services and devices from a variety of protocols including EnOcean, UPnP, and Watteco. The motivation for SPOK is to give the power back to end-users so that they can shape their own smart home at will. This paper reports lessons learned from the methods we have used to validate our hypotheses as well as a number of technical issues concerning development of this type of EUDE. A Video of SPOK in action as of October 2013 is accessible at: http://iihm.imag.fr/demos/appsgate/appsgate2013.mp4

Author Keywords
End-user programming; end-user development; smart home; smart environment; ambient intelligence; ubiquitous computing.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.



The DomiCube. The DomiCube is a home-made device designed by 5 retired seniors as the result of a 3 hour focus group conducted in the Creativity lab of AmiQual4Home (https://amiqual4home.inria.fr) It contains an accelerometer and a gyroscope, and is Bluetooth enabled. It sends events as it is moved including its orientation.

Introduction

SPOK (Simple PrOgramming Kit) is an End-User Development Environment (EUDE) that permits non-specialists to configure, define and control the behavior of their smart home using a pseudo-natural language. With SPOK, inhabitants are not forced to consume the predefined solutions imposed by home automation. Instead, they can configure smart home services by developing programs that fit their needs in an opportunistic manner. The hypothesis is that people are willing to shape their own interactive spaces by coupling smart artifacts, providing new functionalities that were not anticipated by system designers [4,9].

The "Do-It-Yourself Smart Home" is becoming increasingly popular. A number of tools and techniques have been developed to support this view including the Jigsaw editor [7], CAMP [19], iCAP [5], or Newman's work on end-user composition with OSCAR [12]. Commercial home boxes such as the ZipaBox and the Vera, are intended for non specialists. In particular, the Scratch-based programming language [15] of the ZipaBox and the rule-based IFTTT [8] propose an attractive graphics concrete syntax and stylistics.

SPOK has been designed as the result of several analyses of end-users needs [2]. Our objective is to go beyond a proof of concept as well as to go beyond the offer of current commercially available home boxes. The challenge is to have a robust, extensible and flexible system so that effective use in people homes can be performed and evaluated. To reach this goal, a solid infrastructure is needed. In the next section, we describe the implementation of SPOK and its underlying infrastructure followed by the lessons learned from this early experience.

Implementation

SPOK and its companion services (e.g., the context manager) are ApAM-compliant components that are dynamically composed and maintained by the ApAM run time infrastructure [3].

ApAM as the baseline middleware

ApAM (Application Abstract Machine) is a component-oriented middleware that extends OSGi/iPOJO in two ways: (1) developers describe an application by intention using a dedicated language as opposed to explicitly specifying composition of components and bindings at design time; (2) from the abstract description of the application architecture, a concrete architecture is computed and updated incrementally by resolving the dependencies between the components currently available in the execution environment. Due to the incremental and dynamic (just-in-time) construction and maintenance made possible by ApAM, a smart home application is resilient to the opportunistic installation or disappearance of devices, sensors, actuators, and web services.

Overall Software Architecture and Deployment

As shown in Figure 1, a smart home application includes (1) a centralized **Smart Home Server** that can run either on a set top box, a Raspberry Pi, or a mini PC, and (2) a number of **Client Interaction Devices** such as SmartPhones, tablets, robots, or home-made objects such as the DomiCube (see sidebar) used by end-users to interact with the Smart Home. The smart home server is structured as two levels of abstraction: the **Core middleware** – which is domain-independent, and the **Extended middleware** – whose components are designed for a particular class of application domains such as smart homes.

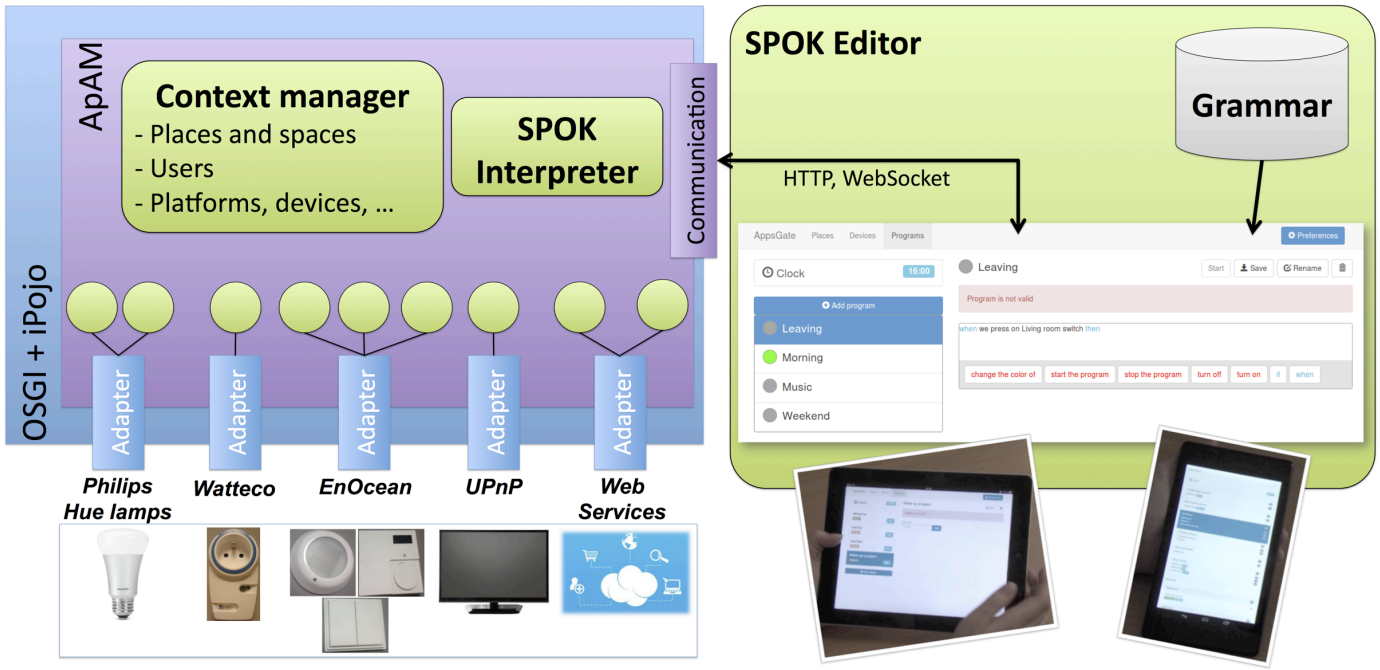


Figure 1. Overall software architecture of the Smart Home middleware on top of ApAM/OSGi-iPOJO and the SPOK editor running on client interaction devices (e.g., tablets, and Smart phones).

The Core middleware of the Smart Home Server
The Core middleware bridges the gap between the external world and the Extended middleware at the appropriate level of abstraction to hide the heterogeneity of the external world. Its functional coverage includes: (1) Technological integration of an heterogeneous set of sensors/actuators network protocols such as EnOcean, UPnP and Philips Hue lights; (2) Uniform representation and management of EnOcean/UPnP/etc. compliant physical devices as well as third parties services, such as Google calendar, email and weather forecast available from the cloud; (3) Communication with client applications; and (4) Automatic support for dynamicity.

1. Adapters as a means to support technological integration. As shown at the bottom of Figure 1, devices from the Physical World are integrated by the way of adapters. Adapters are implemented as OSGi bundles. There is one adapter per type of protocol supported by the Core middleware.

2. "Core world" as a uniform representation of the physical world and of the cloud digital world. Core

World denotes the representation that the core middleware has about the physical world and of the cloud digital world. This representation is expressed in terms of Core objects. Core objects are implemented as ApAM components. For example, a CoreTemperatureSensor is an ApAM component that represents a physical temperature sensor handled by one or more physical sensors network protocols (e.g., the Watteco or the EnOcean sensor network protocols). Similarly, CoreiCal is the abstract representation of any Calendar service made available in the cloud. Any CoreObject may be reused and enriched as an ExtendedObject in the Extended Middleware to fit the purpose of the services, such as the Context Manager, provided by the Extended Middleware. For example, an ExtendedCoreClock has been developed to produce virtual time, flying at some specified speed, so that users can test the execution of their program in the virtual time of their choice.

3. An asynchronous client-server architectural style for supporting communication between the smart home server and client interaction devices. As shown in Figure 1, the **Communication Component**, which is implemented as an OSGi bundle, is the mediating component between the clients and the Smart Home server. It supports web sockets connectors as well as http.

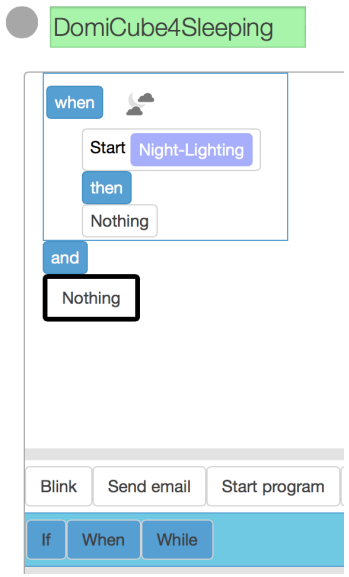
4. ApAM mechanisms as a way to automatically support dynamic environments. Dynamic means that the entities of the physical world, as well as their representation in the Core World, can appear, change or disappear as a result of events that happen in the physical world. ApAM handles these events automatically to ensure that the system is resilient to changes in the physical world. With ApAM there is no need for the developer of the Core middleware to explicitly accommodate changes.

The Extended middleware of the Smart Home Server

The Extended middleware is an extensible set of ApAM-compliant software components designed for facilitating the development of smart environments. In the current implementation, it includes a minimalist context manager and the SPOK interpreter whose role is two-fold: (a) it interprets the abstract syntax tree of the currently active SPOK programs in terms of the API provided by the extended/core objects of the middleware; (b) it provides client applications with contextual information about the state of the smart home (e.g., list of light bulbs that are currently on in the living room) as well as events in which clients have expressed interest (e.g., light bulb L has turned off).

Client interaction devices

In the current implementation, each client interaction device hosts a remote controller that allows users to control the devices, services, and programs in a centralized manner. This provides an alternative to managing multitude of physical remote control devices. The client interaction device also hosts a copy of the SPOK editor, to allow users to build and edit SPOK programs. To maximize portability, all of these software components are developed with HTML5/ JavaScript, and the PhoneGap framework is used to generate hybrid mobile applications for iOS and Android platforms.



The **Smart Keyboard** is updated according to the current entry point as well as to the current state of the home.

In this example, "Nothing", which denotes a placeholder, is the current entry point selected by the user. At the bottom of the screen, the Smart Keyboard shows the actions that make sense for this entry point such as "Send email" or "Start a program", as well as the statements that are syntactically correct in this context (e.g., "If" and "while" statements). Note that only a small part of the keyboard is visible in this picture.

SPOK Key Features

The SPOK language has been designed as a compromise between expressive power and low entry cost. To achieve this, we have defined a powerful language while at the same time hiding complexity by the way of a pseudo-natural concrete syntax and the use of a Smart Keyboard (see Sidebar).

Expressive power

The SPOK language combines rule-based and imperative programming. Typically, a program is a suite of Event-Condition-Action (ECA) rules that are interpreted in parallel. The action part of a rule may include sequential "while" and "if" statements as shown in Figure 2.

In order to support both instantaneous (inchoative) actions and long (durative) actions, conditions may refer to events (e.g., light has been switched off") or to states (e.g., light is off). Given that the underlying middleware is event-driven, references to states are translated by the SPOK interpreter into interest in pairs of events that respectively denote the entry and exit from states.

SPOK supports encapsulation (a program can call another program) as well as parallelism at multiple levels of granularity: several programs can be executed in parallel, and several instructions within a program can be executed in parallel.

Low entry cost

The SPOK grammar is dynamically extended with the grammatical elements provided with new classes of devices. As a result, there is no need for end-users to "hack" the grammar by hand to accommodate a new device. In addition, if a user is not pleased with the vocabulary, SPOK allows the names that serve as terminal symbols to be edited.

Interviews with 17 families, have led us to define a syntax based on natural language [2]. Given that a true natural user interface is not so natural [13], we have opted for the use of a pseudo-natural language with special attention for supporting feedforward [20].

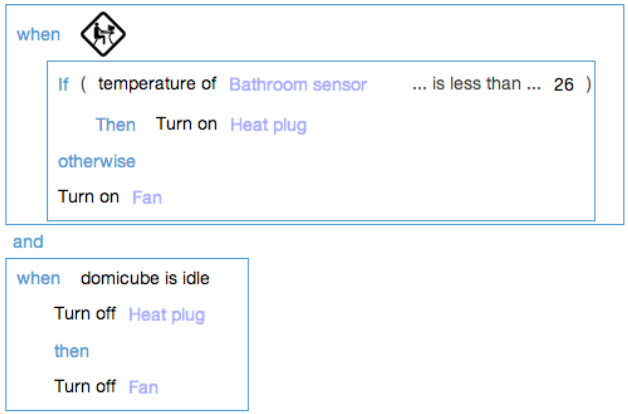


Figure 2. An extract of a program expressed in English pseudo-natural language. This program contains 2 rules (denoted by "when") that are evaluated in parallel. "When the DomiCube is put on the side whose icon shows someone working at a desk, if, at this point, the temperature in the bathroom is below 26°C, then the heater should be turned on, else the fan should be turned on". On the other hand, as soon as "the Domicube is put on its idle side, then turn the heater off followed by turning the fan off".

The SPOK editor supports feedforward for writing programs by providing end-users with a dynamic **Smart Keyboard** that contains the set of correct instructions and references to states and names for the

current entry point (see Sidebar). This is made possible by the dynamic nature of the smart home server.

Early Lessons Related to User studies

To perform end-user studies in the home, we must respect temporal and cognitive constraints of inhabitants, as well as the privacy of their intimate personal space. Used in isolation, interviews, journal studies, and playful cultural probes [1] are unable to comply with such requirements. This problem may be avoided by a complementary combination of multiple techniques [4, 6, 11, 16]. Among these techniques, a particularly effective approach is to visit the home with the inhabitants. The challenge is to do this without having observers intrude on the inhabitant's personal space.

Figure 3. An example of two play cards association presented on a tablet and its related question: "What services, whether it be useful or not, could results from a communication or cooperation between your washing machine and your TV?"

We have found [2] that a particularly effective means to visit a home, is to have people take snapshots of their personal space and objects as shown in Figure 3. These photos are then used as material for informal discussion and interviews as well as playful cultural probes. Unlike "follow-me" visits, snapshots allow inhabitants to present their personal space in a way that inspires subject involvement and creativity without intrusion on privacy or hygiene of personal spaces.

The details of our field study and findings are presented in [2]. In short, we have found a number of facts that are consistent with results reported in prior literature. This includes the existence of key moments organized in well-established procedures (e.g., "wake-up time") as exemplified by this sentence: "In the morning, picking up the towel after the shower, should trigger the coffee machine so that coffee would be ready just in time, at the right temperature, along with the radio turned on in the kitchen broadcasting the news using the appropriate sound level". In addition to the desire for chaining services, our data from the association game (as illustrated in Figure 3) shows that family members are prone to envision new services when coupling involves one "communicating" object, or one "programmable" object. In particular, service improvement is required for appliances that are not sufficiently equipped by coupling them with additional input and output facilities such as those of the TV set (cf. Figure 3): "I should be able to control the washing machine located in the basement using the TV".

This field study provided us with the appropriate arguments for developing SPOK.

Early Lessons Related to Technical Issues

There is currently no reference middleware for smart homes. The implementation of a tool for EUDE requires the existence of the "appropriate" run time infrastructure in a jungle of middleware. From our experience with the development of SPOK, "appropriate" means the capacity to satisfy two types of requirements: intrinsic functional power and accessibility to programmers.

Intrinsic Functional Power
The opportunistic arrival/departure of devices and services requires the dynamic discovery, (re)composition, and deployment of software components. From this perspective, OSGi provides a good baseline. ApAM, which hides the complexity of OSGi and performs just-in-time component composition and deployment in a transparent way for the programmer, provides the appropriate support.

Given the lack of a reference middleware for smart homes, interoperability with other infrastructures is a clear advantage over closed mono-technological spaces such as Apple HomeKit or .net-based HomeOS [18]. Openness, not only avoids technology lock-in, but makes it possible to take advantage of the best of multiple worlds. For example, being interoperable with a popular middleware such as ROS (Robot Operating System) [17], will allow developers to extend smart homes with the capabilities of multiform (social) robots.

The current lack of a *de facto* standard to abstract away the physical world is strongly related to the lack of interoperability between devices and services. Most middleware for the home include this "core level". Unfortunately, there is no consensus on the right model. The requirement for both synchronous and asynchronous communication including http and REST API to communicate with clients is also a problem.

Accessibility to Programmers
As researchers, we need robust open source middleware with a large community of developers and users to support the evolution and perennity of our developments. OpenHAB [14], which has been chosen by Eclipse to become the Eclipse SmartHome, is a good option. On the other hand, in its current form, OpenHAB does not support the dynamicity mentioned above.

Programmers of the domain-dependent level generally seek a low entry cost. OSGi, is not a good option for this. In addition, they want to use their favorite programming language, which is not necessarily Java.

Conclusion
Due to the lack of a reference middleware for smart homes, we have spent a significant engineering effort in the development of the core middleware at the expense of our research agenda on SPOK. As a result, a debugger [10], coupled with a home simulator, is still under development. Similarly, although we have performed minimal deployments in the homes of several members of the development team, experiments in the wild remain to be performed.

With regard to field studies, using inhabitant "snapshots" as a cultural probe of home environments has multiple advantages: (1) It serves as an ice breaking between the family members and the experimental team; (2) Family members "reveal their house" naturally while the experimenters do not intrude their private spaces. (3) Family members get truly involved and become intrigued by what will come next. (4) As opposed to playful probing proposed by R. Bernhaupt [1], snapshots use images of personal objects, rather than generic entities. This increases the interest and imagination of the participants while improving the meaningfulness of the data collected.

Acknowledgements
This work is supported by the European AppsGate CA110 project. The DomiCube was built in the Creativity Lab of the EquipEx AmiQual4Home, ANR-11-EQPX-00. We wish to thank C. Lenoir, N. Mandran, and

C. Roux for the user studies, J.R. Courtois, R. Dautriche, and K. Pethoukov for the technical development of SPOK, and J. Estublier, T. Flury, C. Gérard, and G. Vega for the development of ApAM and of the Core level middleware of the Home Server.

References

[1] Bernhaupt, R., Weiss, A., Obrist, M. & Tscheligi, M. (2007) Playful Probing: Making Probing more Fun. In *INTERACT 2007, Springer LNCS*, 606-619.

[2] Coutaz, J. et al. (2010) DisQo : A user needs analysis method for smart home. In *Proc. NordiCHI 2010 International Conference*, ACM, 615-618.

[3] Damou, E. (2013) *ApAM : un environnement pour le développement et l'exécution d'applications ubiquitaires*. Thèse de doctorat de l'Université de Grenoble, spécialité informatique.

[4] Davidoff, S., Lee, M.K., Yiu, C. Zimmerman, J. & Dey, A. (2006) Principles of Smart Home Control. *In Ubicomp 2006, LNCS 4206,* Springer, 19-34.

[5] Dey, A., Sohn, T., Streng, S. & Kodama, J. (2006) iCAP: Interactive prototyping of context-aware applications. In *Pervasive 2006,* Springer, 254-271.

[6] Holloway, S., Stovall, D., & Julien, C. (2009). *What Users Want from Smart Environments*. Technical Report, UT-EDGE-2009-008.

[7] Humble, J., Crabtree, A., Hemmings, T., Akesson, K.P., Koleva, B., Rodden, T. & Hansson, P. (2003) "Playing with the bits" user-configuration of ubiquitous domestic environments. In *Ubicomp 2003*, 256-263.

[8] IFTTT. https://ifttt.com.

[9] Intille, S. (2002). Designing a home of the future. *Pervasive Computing*, IEEE, 76-82.

[10] Ko, A. J. and Myers, B. A. (2008) Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior. In proc. *International Conference on Software Engineering (ICSE)*, ACM, 76-82.

[11] Mennicken, S. and Huang, E. M. (2012) Hacking the Natural Habitat: an in-the-wild study of smart homes, their development, and the people who live in them. In proc. *Pervasive 2012*, IEEE, 143-160.

[12] Newman, M.W., Elliott, A. & Smith, T.F. (2008) Providing an integrated user experience of networked media, devices, and services through end-user composition. In proc. *Pervasive 2008*, IEEE, 213-227.

[13] Norman, D. (2010) Natural User Interfaces are not Natural. In *Interactions, Vol 17, No 3,* ACM, 6-10.

[14] openHAB. www.openhab.org.

[15] Resnik, M., Maloney, J., Monroy-Hernández, A., et al. (2009) Scratch: Programming for all. *Communication of the ACM,* 52, 60–67.

[16] Rode, J.A. & Toye, E.F. & Blackwell, A.F (2005) The domestic economy: A broader unit of analysis for end user programming. In proc. *Conference on Human Factors in Computing Systems (CHI 2005)*, 1757-1760.

[17] ROS. Robot Operating System, http://www.ros.org.

[18] Rosen, N., Sattar, R., Linderman, R. W., Simha, R., & Narahari, B. (2004). HomeOS: Context-Aware Home Connectivity. In proc. *International Conference on Wireless Networks*, CSREA Press, 739-744.

[19] Truong, K.N., Huang, E.M. & Abowd, G. (2004) CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In *Ubicomp 2004, Springer*, 143-160.

Vermeulen, J., Luyten, K., van den Hoven, E., & Coninx, K. (2013). Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity. In Proc. *Conference on Human Factors in Computing Systems (CHI 2013)*, ACM, 1931-1940.