Taking Smart Space Users Into the Development Loop

Marc-Oliver Pahl
TU München
Boltzmannstr. 3
85748 Garching, Germany
pahl@net.in.tum.de

Georg Carle
TU München
Boltzmannstr. 3
85748 Garching, Germany
carle@net.in.tum.de

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2215-7/13/09...$15.00.

http://dx.doi.org/10.1145/2494091.2497320

Abstract

Smart spaces need driver services to connect accessed hardware and orchestration services to realize scenarios. There is a problem of scale in software development for smart spaces because it is done by few. It is also problematic that those few decide about what is supported and developed. We propose to provide users with tools for community based development of driver and orchestration services. We analyze the requirements for a middleware framework to allow distributed development. We present necessary extensions that promote community based development: (1) a repository for interface definitions, (2) App Store and App Manager, and (3) multi-dimensional ratings.
Finally we present how smart space software development can be facilitated using our Distributed Smart Space Orchestration System (DS2OS).

Author Keywords

community based development; crowdsourcing; ontology; heterogeneity; driver; orchestration; smart space

ACM Classification Keywords

D.2.11 [Software Architectures]: Domain-specific architectures; K.4.3 [Organizational Impacts]: Automation; H.1.2 [User/Machine Systems]: Human factors

General Terms

Economics, Design, Human Factors, Management

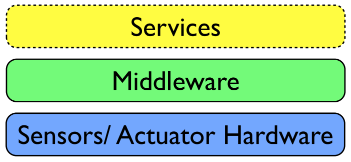

Figure 1: Middleware decouples the hardware from the orchestration logic.

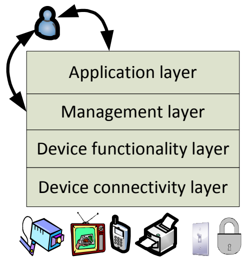

Figure 2: HomeOS [6].

Introduction

Smart spaces have a fundamental problem: who programs the software? The heterogeneous hardware devices that make spaces smart need software **driver services** to become *interoperable* with middleware [18] that is used. Scenarios from areas like user convenience, security and safety, support for elderly or disabled, or energy saving [10] require the development of **orchestration services** that implement the *scenario-specific workflows*.

Today only geeks are able to make and maintain Do-It-Yourself (DIY) installations [4]. Even after the hardware is installed (by professionals or DIY), maintaining and extending the software of a smart space remain difficult. Skills and enthusiasm at the user's side often exist [4, 14, 13], but the support for developers by middleware frameworks is weak or missing.

After introducing three current middleware designs for smart spaces we identify requirements for community based software development for smart spaces. We present three core concepts that enable the user community to develop and share drivers and orchestration services: (1) a **repository for interface definitions**, (2) an **App Store** and an **App Manager**, and (3) **multi-dimensional ratings**. Finally we show how our Distributed Smart Space Orchestration framework facilitates the creation of smart space services.

Middleware for Smart Spaces

Smart spaces should be orchestrated via software [7, 5, 15, 3, 18]. Software based orchestration allows flexible shared use of hardware devices [10] by reconfiguring and deploying services dynamically for realizing different scenarios.

Middleware helps realizing software orchestration by providing an *abstract interface* to the hardware (see Fig. 1), and by *facilitating* software development.
Abstract interfaces to the functionality of the hardware combined with *discovery mechanisms* allow decoupling software from specific hardware devices. Device functionality (e.g. is-a lamp) can be programmed and found based on abstract interfaces instead of concrete device addresses. This is an important step towards distributed software development, enabling services to be developed in one smart space and run in other smart spaces (portability).

To take users into the development loop, programming must be easy. Common middleware facilitates software development process by introducing *consistent interfaces* on top of heterogeneous hardware devices, and by offering *common functionality* for services which thereby does not have to be reprogrammed for each service, as it is the case for access control for instance.

Subsequently, three middleware designs are presented with focus on the *abstraction* they use, the *support* they provide for developers, and how *devices are integrated* into the platform. Device integration is especially important as devices that can be used for software orchestration define the possibilities of the system, because devices are the interface between the software and the physical world.

Microsoft – HomeOS

At Microsoft *HomeOS* is designed [7, 6]. Orchestration services run on a central computer using local device drivers as interfaces to distributed devices.

The HomeOS middleware uses so-called *roles* as abstraction. A *role* is a collection of **method signatures**. *Roles* are intended for device drivers [6]. To implement a certain *role*, a device driver has to offer the methods specified in its *role* (see Device Functionality Layer in Fig. 2).

The system offers functionality to control and mediate access to devices and to add and remove Applications and devices. Services are implemented as *.net* programs.

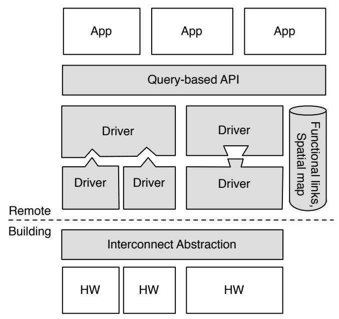

Figure 3: BAS [12].

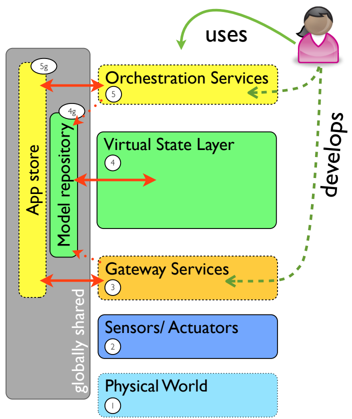

Figure 4: DS2OS [17]

Drivers and orchestration services offer the same API. Implementations for specific protocols (e.g. z-wave) can be used as base for new device drivers.

Berkeley – Building Application Stack (BAS)
The University of California in Berkeley develops the Building Application Stack (BAS) [12, 5].

BAS [12] can be distributed. It realizes service portability via abstract driver interfaces to devices. The Building Operating System Services (BOSS) [5] provide supplementary functionality for easing service development.

Like in HomeOS the system abstraction is **method based**. So-called *classes* bundle the methods that a driver for a certain device type must offer. Coupling of orchestration services is not described.

The framework provides functionality for transactions, authorization, and time series. Services for BAS are written in Python.

As in HomeOS the development of device drivers is structured in different hierarchical steps (see Fig. 3). Only the highest level of abstraction is meant to be used by service programmers.
BAS uses sMAP[1] as low-level *driver* to sensors and actuators. sMAP is an open-source framework that provides a web-based RESTful Application Programming Interface (API) as abstraction on top of the heterogeneous device protocols of the sensors and actuators.

TU München – Virtual State Layer (VSL)
The Virtual State Layer (VSL) is developed at the Technische Universität München [17]. The VSL is the middleware of the Distributed Smart Space Orchestration System (DS2OS). DS2OS realizes a distributed Service Oriented Architecture (SOA) for smart spaces.

[1]http://code.google.com/p/smap-data/

While the other middleware designs use collections of method signatures as abstract interface the VSL uses *hierarchically structured tuples*. The VSL tuples are basically key-value pairs that form a tree. The tree representations of the abstract interfaces of services are called *models*.

VSL offers functionality for communication, access control, transactions, and persistence. Information tuples are stored within VSL independent of the connected services or devices. The VSL is not only the information broker between services but an active component in DS2OS with the persistence it offers.

The communication of VSL is not based on remote procedure calls but on a distributed tuple space [8] (see Fig. 4). This allows to use a **fixed set of methods on variable data** while the other middleware designs have variable methods and variable data.

Unlike other frameworks DS2OS does *not differentiate between driver and orchestration services*. Both are *regular services*. Driver functionality is *divided* into a part that is is directly interfacing a device's protocol, and a part that provides higher level abstractions. Different to the other designs, the information between the different driver stages is brokered over the middleware. This gives *additional flexibility* and *enables distributed development* of driver components.

Requirements for Community Based Development of Services for Smart Spaces

To allow community based development services must be portable. Portability leads to the following requirements on a middleware design:

- **Abstract interfaces on top of the hardware** to allow *portability of services*.
  The *roles*, *classes*, and *models* of the presented middleware designs fulfill this property.

- **Published description of service interfaces** to allow the *discovery and (re)use of services*.
  This is possible for all named middleware designs.
- Support of **dynamic binding** / lose coupling of services by the middleware to allow the *provisioning of services at runtime*.
  This feature often comes with the availability of abstract interfaces and the support to discover functionality as such features automatically decouple the different software components of a system.

All presented middleware designs fulfill the requirements for community based service development.

Necessary Extensions for Community Based Development

To promote the community based development of smart space software, the following extensions to the presented frameworks are necessary.

A Repository for Interface Definitions

To allow **distributed development of components** and to promote **convergence**, it is essential to share the abstract interface definitions (*roles / classes / models*).

Only known abstractions can be used by programmers. Especially with the high heterogeneity of smart devices it is unlikely that a programmer of a software will possess all devices she develops software for on the one hand.
On the other hand, driver service developers can only reuse the abstract interfaces they know for the device types they want to support.

Therefore it is highly beneficial to create a global repository that contains the abstract interface definitions of the services available for smart spaces.

HomeOS has a *HomeStore* [6] that provides services and meta data about dependencies to end users. It could be extended to provide not only information for end users but also the proposed interface definitions (roles) for developers. BAS [12] does not mention a global repository. DS2OS has a global *Model Repository* for interface definitions, which serves the described purpose.

A Global App Store and a Local App Manager

People enjoy developing software [14, 19] and like sharing it. The open-source movement [13] and the App economy [1, 16, 20] are successful examples for community based software development.

The success of the App Store first for smartphones and then also for PCs shows that it helps to have a convenient and centralized distribution channel for software.

To allow the **sharing of software services**, a global software repository (App Store) is most suitable. It should provide developers with functionality to (1) share their software, (2) distribute updates, (3) get social fame, (4) earn money, (5) receive feedback, and (6) run tests.

Developers can offer, manage, and update their software over an App Store, which provides a channel to the customers. During development the App Store can be used for controlled beta tests. At normal operation of a service the App Store can be used for **automatic deployment of updates and as feedback channel** for receiving encouraging comments, fame, and money [16].

End users benefit from an App Store by having a broad base of software services (e.g. device drivers, orchestration services) at hand. With the presented sharing mechanism, skilled users can contribute the extensions they developed for their smart spaces to the community. This allows users that are unable or unwilling to develop software components for smart spaces to benefit by using the software from the community App Store.

Sharing services results in a big amount of Application scenarios and supported devices in short time. This attracts users and new developers (see App economy for smart phones [1, 16, 20]). With community based development of driver services the proposed approach *solves the scalability problem* of the high number of heterogeneous hardware devices available and already deployed.

To **deploy and manage the software** in a smart space instance, a local App Manager is needed. It *facilitates the installation* of software services (e.g. by automating it) and allows *automated updates* (e.g. of device drivers). The local counterparts of an App Store in the smartphone ecosystem, in operating systems, or in software itself exemplify how a local App Manager works with features like automatic update checks and installations. Different to the smartphone domain, the heterogeneity of smart spaces brings the additional requirement that an App Manager for smart spaces must check dependencies on required resources (including available hardware devices for sensing and actuating) before installing software.

The HomeStore of HomeOS [7, 6] distributes software. To communicate the dependencies, each service brings a *manifest* describing which *roles* must be present inside a space to run the service. HomeStore also suggests hardware to users which they need in order to run a service and have a certain functionality in their space. BAS does not describe the functionality of a software repository. For DS2OS we are currently developing a local App Manager and an App Store as described, including automated provisioning of services (e.g. device driver services), see 5g in Fig. 4.

The local App Manager of DS2OS takes care of **automated updates** of the local services. It **collects the runtime behavior** (e.g. errors) of a service and **reports** this information in a privacy conform way to the App Store. From there other potential users and the developers can be informed leading to a continuous evaluation of the runtime behavior of services. The Approach is suitable to **identify bugs quickly**. To our knowledge such a mechanism does not exist in a fully automated way today. Finally the local App Manager allows **remote management**, e.g. if some family members (e.g. children) are more experienced and want to support the others (e.g. parents).

Different to the other middleware designs, DS2OS *drivers are regular services* allowing developers to *benefit from the full set of DS2OS features*. The approach allows the App Store and the local App Manager to *handle drivers like any other service*. The service manager can provision drivers without user intervention if devices within a space can be discovered with protocol mechanisms and drivers are available from the App Store.

With the drivers being regular services and the *automated provisioning* via the local App Manager users automatically and immediately benefit from drivers that others developed. For DS2OS we implemented a scenario to show that this is feasible using managed network devices communicating over SNMP. Especially in fully automated processes quality assurance is important. A solution for community based quality assurance is introduced next.

Multi-Dimensional Ratings

A fundamental problem of smart spaces today is the heterogeneity in the hardware and the resulting heterogeneity in the software interfaces and the protocols involved. As described at the end of the App Store section an important task is to develop device driver services. As mentioned in the middleware section, such drivers offer an abstract interface to a device family (e.g. lamps).

When distributing the software development it is natural that without arbitration, diversity will emerge. If each device driver service for a member of a device family offers a different abstract interface, the heterogeneity problem is only shifted to a higher level of abstraction, preventing service portability again. The necessity for convergence Applies to services that offer abstract information as well. The DS2OS architecture makes use of such services to modularize services and to extend the functionality offered by the framework which requires to converge their interfaces too. To promote **convergence** in the abstract service interfaces and to assure good **service quality**, we propose a community based rating process for the interface repository and the App Store. Our proposal is based on explicit and implicit ratings of abstract interfaces and service implementations. Our proposal scales better and is more democratic than Apples individual App software checks[2] as it is not company controlled but based on the user community. At the same time our proposal uses more channels and thereby evaluation criteria than Google's approach[3] with automated tests. Both can be added to our proposed architecture.

We introduce different ratings for **abstractions** in the interface repository, and for **services** in the App Store. Explicit (subjective) feedback is collected from the community in form of numeric ratings (e.g. 1=worst..5=best). Implicit (more objective) feedback is generated by correlating data from the interface repository with the App Store and by collecting anonymous feedback from local App Managers.

We introduce the following ratings for models:
(1) The **use of abstractions** available in the interface repository is automatically correlated with the software in the App Store to identify the objective popularity of an abstraction. The popularity depends on the number of services using the abstraction and the popularity of those services. Knowing the popularity of an abstract interface helps developers to use popular abstractions, as they are supposed to have the biggest device installation base making new services more attractive. In turn developers of device drivers can use the abstractions that are currently most often used by orchestration services, making their new driver more attractive. (2) Developers can manually rate the **subjective popularity of abstractions** based on the usability and the quality of the abstract interfaces.

The knowledge of the popularity of a model leads to natural selection. Developers will support more popular models as they seem to be better and as they are spread wider. Driver developers will use the more popular models as they want to be compatible with the popular services. This automatic feedback loop leads to convergence of models.

For services we introduce the following metrics:
(3) The recent **amount of downloads** of a service from the App Store is monitored. (4) The **amount of currently running copies** of a service is monitored via anonymous feedback from the local App Managers. (5) The **amount of service crashes** and the **relative uptime** of services are monitored via anonymous feedback from the local App Managers. (6) End users can **manually rate services**.

The knowledge of the popularity of a service leads to natural selection. The metrics (3-6) will be used as quality indicator by people browsing the App store for interesting services. The error metric (5) helps users to identify stable services and developers to improve their services.

All proposed metrics must be publicly available (e.g. via pages of the App store and the model store). This way a **natural selection of models and services** will happen.

Support of DS2OS for Developing Software for Smart Spaces

DS2OS is designed to facilitate development of software for smart spaces. The interaction paradigm of the VSL

[2] http://store.apple.com/
[3] http://play.google.com/

middleware as tuple space with a **fixed set of methods** and **variable data** facilitates the creation of services significantly as it separates data from functionality [9]. The separation reduces complexity and makes the system better extensible. Fixed methods can be learned and memorized better than the method based interfaces often used by other middleware designs. Additionally the fixed method set can easily be transferred into symbols for *visual programming*. A visual programming interface facilitates programming and enables it for beginners [11].

The abstract interface of a DS2OS service is a tree of nodes that is called *model*. Thus the **abstract interface** of a DS2OS device driver service describes the **properties of the physical device** it connects to the middleware. Creating a list of properties as abstract interface is more at hand than creating method signatures.

The **model repository is the ontology** [2] of a DS2OS space. Thus tools for context modeling can be used to create the abstract interface definitions. Especially for beginning programmers such an abstraction is **closer to reality** than method signatures.

Models can inherit from other *models*. This allows to extend models easily (e.g. by creating a *model* "specialLamp" that is based on the basic "lamp") and helps to converge models. An extended model offers the nodes (interface) of the model it extends and contains the type of the original *model* for the discovery of the included parent functionality. Creating a new *model* that is based on an existing *model* facilitates the creation of abstract interfaces.

Additional to access control, transactions, publish-subscribe mechanisms, and communication the VSL offers **persistence** for software developers. In combination with the fixed set of methods and the ontology abstraction software developers can entirely focus on the logic they want to realize in their orchestration services [9].

Handling all services equally allows the reuse of services for other services. This helps to **modularize functionality** and to **chain services** for raising the level of abstraction (e.g. a service could deduce that it is day and offering that information for all services in a space). As all services are equal, the functionality software programmers can build upon can easily be extended.

Conclusions

We presented DS2OS, an architecture for community based software development for smart spaces. The paper identified general requirements on smart space orchestration frameworks to enable community based development. Three extensions to promote community based development were presented: (1) a repository for interface definitions, (2) App Store and App Manager, and (3) multi-dimensional ratings. We showed how DS2OS exceeds the developer support of other orchestration frameworks for smart spaces.

We are currently running a study on the developer friendliness of DS2OS. The extensions that we propose in this paper are currently getting implemented.

Encouraged by recent surveys [4, 14] we believe that realizing our proposal at large scale will lead to community based software service development for smart spaces. The proposed features allow non developers to benefit. We believe that the critical mass of developers already exists and that only the proposed infrastructure is missing today.

Community based development may automatically solve the scaling problem of driver development for the plethora of smart devices and their protocols. At the same time it empowers the community to choose the devices that should be supported and to define which orchestration services are considered interesting.

The resulting availability of smart spaces in more households, orchestration services that realize attractive scenarios, and driver services that bring added value to the existing hardware may raise customer interests in smart space technology possibly opening a huge market [4, 7, 14].

References

[1] Anthes, G. Invasion of the mobile apps. *Communications of the ACM 54*, 9 (2011), 16–18.

[2] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing 6*, 2 (Apr. 2010), 161–180.

[3] Bourcier, J., Diaconescu, A., Lalanda, P., and McCann, J. A. AutoHome: An Autonomic Management Framework for Pervasive Home Applications. *Transactions on Autonomous and Adaptive Systems 6*, 1 (Feb. 2011).

[4] Brush, A. J. B., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., and Dixon, C. Home Automation in the Wild: Challenges and Opportunities. In *CHI 2011*, ACM Press (2011), 2115.

[5] Dawson-Haggerty, S., Krioukov, A., Taneja, J., Karandikar, S., Fierro, G., Kitaev, N., and Culler, D. BOSS: Building Operating System Services. *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2013).

[6] Dixon, C., Mahajan, R., Agarwal, S., Brush, A. J., Lee, B., Saroiu, S., and Bahl, P. An operating system for the home. In *NSDI'12: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (Apr. 2012).

[7] Dixon, C., Mahajan, R., Agarwal, S., Brush, A. J., Lee, B., Saroiu, S., and Bahl, V. The Home Needs an Operating System (and an App Store). In *the Ninth ACM SIGCOMM Workshop*, ACM Press (2010), 1–6.

[8] Gelernter, D. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)* (1985).

[9] Grimm, R., Davis, J., and Hendrickson, B. Systems Directions for Pervasive Computing. *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems* (2001).

[10] Holroyd, P., Watten, P., and Newbury, P. Why Is My Home Not Smart? In *Aging Friendly Technology for Health and Independence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, 53–59.

[11] Kaucic, B., and Asic, T. Improving introductory programming with Scratch? In *MIPRO, 2011 Proceedings of the 34th International Convention* (2011), 1095–1100.

[12] Krioukov, A., Fierro, G., Kitaev, N., and Culler, D. Building application stack (BAS). In *BuildSys 2012* (2012), 72.

[13] Lerner, J., and Tirole, J. Some Simple Economics of Open Source. *The Journal of Industrial Economics 50*, 2 (Mar. 2003), 197–234.

[14] Mennicken, S., and Huang, E. M. Hacking the natural habitat: an in-the-wild study of smart homes, their development, and the people who live in them. In *Pervasive'12: Proceedings of the 10th international conference on Pervasive Computing*, Springer-Verlag (June 2012).

[15] Miori, V., Russo, D., and Aliberti, M. Domotic technologies incompatibility becomes user transparent. *Communications of the ACM 53*, 1 (Jan. 2010), 153.

[16] P, A., Matos, Christina, C, A., Vanessa, Michael, and Stijn. Developer Economics 2012. Tech. rep., London, June 2012.

[17] Pahl, M.-O., and Carle, G. The Missing Layer - Virtualizing Smart Spaces. In *10th IEEE International Workshop on Managing Ubiquitous Communications and Services 2013 (MUCS 2013)* (2013), 139–144.

[18] Raychoudhury, V., Cao, J., Kumar, M., and Zhang, D. Middleware for pervasive computing: A survey. *Pervasive and Mobile Computing* (Sept. 2012).

[19] Takayama, L., Pantofaru, C., Robson, D., Soto, B., and Barry, M. Making technology homey: finding sources of satisfaction and meaning in home automation. In *UbiComp '12: Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (Sept. 2012).

[20] Vakulenko, M., Schuermans, S., Constantinou, A., and Kapetanakis, M. Mobile Platforms: The Clash of Ecosystems. Tech. rep., Nov. 2011.