Requirements for Smart Home Applications and Realization with WS4D-PipesBox

Christian Beckel, Heinz Serfas
Robert Bosch GmbH
Corporate Sector Research
and Advance Engineering
Postfach 30 02 40
70442 Stuttgart, Germany
{christian.beckel, fixed-term.heinz.serfas}
@de.bosch.com

Elmar Zeeb, Guido Moritz, Frank Golatowski,
Dirk Timmermann
Institute of Applied Microelectronics
and Computer Engineering
University of Rostock, 18057 Rostock, Germany
{elmar.zeeb, guido.moritz, frank.golatowski,
dirk.timmermann}@uni-rostock.de

Abstract

The increasing level of device connectivity in today's homes and buildings enables numerous opportunities for home owners, building managers, device manufacturers and solution providers. Standardized communication protocols (e.g., ZigBee, Bluetooth) provide physical connectivity and thus serve as a basis for smart home applications. However, beyond physical connectivity, real interoperability to effectively develop such applications requires additional efforts. It requires harmonizing multiple protocol standards, dealing with a heterogeneous device landscape, different data formats, managing resource constraints of devices and providing means to react quickly when devices and applications leave or join the system. We propose to address these application-specific requirements in multiple layers of abstraction. Our research has shown that each layer is used by different types of developers (e.g., device supplier, service provider, home owner) with specific tool support. The paper provides a classification and detailed analysis of requirements that have to be addressed in order to enable application development in smart homes. It further proposes and analyzes WS4D-PipesBox, a multi-layer framework, to illustrate how applications could be developed using multiple layers of abstraction.

1. Introduction

Interconnected devices enable to intelligently monitor and control smart homes in a future Internet of Things. Energy saving applications, for example, control indoor climate and electricity usage by employing context information to switch off appliances (e.g., lights, computers), reduce room temperature, close windows, or stop warm water circulation. In addition to smart energy usage, applications from domains security, comfort and telehealth will be prevalent in smart homes and buildings.

In contrast to mobile phones smart home systems exhibit a much higher level of heterogeneity in terms of hardware setup and user needs. Thus off-the-shelf solutions which consist of a set of devices and preconfigured software bundles will satisfy customer needs only to a very limited extent. Alternatively customers could engage installers to deploy and configure home automation systems and tailor them to their individual needs. However, employing installers is expensive, which would reduce the value of home automation systems for customers and thus impede their penetration. Calling the installer again when users buy additional hardware or change their preferences further limits the customer group to a small set of rich people.

Both off-the-shelf solutions and solutions that are complex to install fail to address the needs of the vast majority of households adequately. We propagate that customers should be able to individually create or modify applications that are tailored to their homes. This allows more adapted and therefore more suitable configurations by taking personal habits and preferences into account. It further removes the cost barrier to call the installer each time modifications must be performed. The process of creating and adapting applications by end-users without necessarily understanding the underlying technological details is what we call *composition*. This concept of creating applications from basic building blocks is widely adopted in the enterprise IT domain and for building management systems. However, existing tools are either too complex to learn and require specific training. They also often focus on a specific application domain (e.g., KNX for home automation) or technology (e.g., BPEL with Web Services), which limits development of heterogeneous applications. Applications from all home automation domains such as *comfort*, *security*, *safety*, and *health care* would all benefit from composition mechanisms. Comfort and health care applications for example must match individual pref-

erences and patients. Security applications highly depend on the items to be protected and on devices which are available. Energy saving applications must adhere to a trade-off between energy savings and potential limitations in comfort.

The following motivating examples illustrate the benefits of a home automation solution in the energy domain. They also reflect that real-life scenarios are too specific for a off-the-shelf solution because they intervene with habits and preferences of home owners. In the first example scenario user preferences derived from a home owner's family calendar or based on pre-defined rules provide a coarse-grained context. This context is augmented with sensor information. Occupancy information, for example, allows for a more fine-grained control of the building and individual rooms. Finally, calling Internet services further contributes by being able to switch appliances on and off based on current energy costs or the local weather forecast. In addition the weather forecast also enables a more accurate planning of a building's energy demand and supply. The second example achieves to increase energy awareness by visualizing energy consumption to users, for example by showing it on a TV. It motivates users to change their behavior and assists them to detect and replace inefficient appliances. Performing analysis on measurements over a certain time frame and putting consumption patterns in relation to statistical data (e.g., average consumption in current season, house calendar) further identifies inefficient appliance settings. Giving users control over such applications by letting them compose individual components on a high level of abstraction makes these specific applications both cost-effective and accepted by users.

However, letting ordinary users compose applications for smart homes involves a large set of requirements, which must be analyzed thoroughly. As an example, users are usually not educated in programming, thus the composition method must be simple to learn and apply. Next, affecting the physical world leads to safety-challenges that are not as relevant in pure software-based systems.

This paper provides an extensive list of requirements for composition of smart home applications and proposes the software framework *WS4D-PipesBox*, which claims to adequately address these requirements. The analysis of *WS4D-PipesBox* shows that different sets of requirements are best addressed on different levels of abstraction, each of which can be performed by a different class of users. This allows for example driver developers to enable support for a wide range of devices. Installers can initially set up, install, and configure a home automation system. Home owners are able to extend the system or change the individual configuration on a high level. The next chapter discusses requirements for composition of applications in smart homes. Many of them differ from established enterprise integration mechanisms. Chapter 3 presents related work covering both standardization efforts and research activities. Chapter 4 provides a generic framework which addresses the requirements using the multi layer approach. Finally, the paper concludes with a discussion on how well *WS4D-PipesBox* fulfills the requirements and provides future research challenges.

2. Requirement Analysis

This chapter categorizes requirements for creating applications in smart homes. The list of requirements provides guidance for tool developers from an industry perspective. A composition tool should either fulfill these requirements inherently or provide means to solution developers (e.g., installers, home owners) to cover relevant aspects with little effort. The requirements are clustered in seven categories, each of which consists of three to five requirements.

2.1. Simplicity

Simplicity describes the complexity of application development. It involves the interaction between the system and the application developer.

- *Learning*: Targeting usually untrained home end-users the composition tool must be easy to learn and simple to use.
- *Building/Changing*: Experienced or trained users should be able to quickly develop or modify even complex applications.
- *Levels of abstraction*: Providing multiple layers of abstraction allows to hide implementation details to end users and to expose them to more advanced developers.

2.2. Modeling

This category deals with requirements that affect the way the smart home applications can be modeled.

- *Eventing*: Applications in smart homes are highly event-driven. This is due to domain characteristics as well as resource and energy constraints of devices. Thus it should be possible to model fine-grained event management (e.g., subscribe, unsubscribe) and event delivery. It should further be possible to model event management, to deal with both synchronous and asynchronous events, and to handle events with defined and undefined order.
- *Expressiveness*: Smart home applications combine information from multiple domains (e.g., health care, security). To make creation of such applications efficient, application developers should be limited in their capabilities to some extent. However, the challenge is to still provide the expressiveness that is needed to develop powerful domain-specific and cross-domain applications.
- *Statefulness*: Modeling states of the complete environment and transitions between states is closely related to state-based devices in the home domain. This results in different behavior of a function with respect

to a system's state (e.g., when logged in or not logged in).

2.3. Time

The ability to impose timing constraints on the system is crucial for two reasons. First, smart home applications affect the real world. Second, applications interact with resource-constrained devices which exhibit limited availability and varying delays. This distinction between real world data timing and communication timing may significantly impact fulfilling the requirements in complex scenarios. For example, the age of a sensor reading may include the real world time of the measurement as well as the time of transporting the data from the source to the sink.

- *Hard real-time*: A system which supports hard real-time guarantees that a certain action is performed within a given time frame. Smart home developers can specify this time frame in application development. In near term we expect no use case that requires actual hard real-time.
- *Soft real-time*: In contrast to hard real-time, missing a time frame in a soft real-time system is not considered as an error but a quality problem. Over time, if soft real-time deadlines are missed more often, system acceptance suffers.
- *Age*: In systems with energy-constrained devices caching mechanisms are used to reduce energy consumption. Providing means to the developer to specify a minimum or maximum age of sensor readings is required (e.g., a heating device which uses room temperature measurements).
- *Synchrony*: Performing actions synchronously (e.g., using a checkpoint-based approach) or intentionally asynchronously allows the developer to specify that events start or end at the same time (e.g., lights are switched on at the same time or avoiding all devices to be switched on at the same time to prevent from peaks in the power supply system). This requirement mostly addresses quality (acceptance) of the system.
- *Periods*: For periodical actions application developers must be able to specify both the period of events and a maximum jitter each event may have.

2.4. Mobility

Mobility includes both mobile devices and changes in the system (e.g., devices and services leave or join the system).

- *Discovery*: Discovery enables detection and integration of devices statically during design time or dynamically during runtime. In case of a repository, devices are located based on a match between their capabilities and the user's preferences.
- *Device Disappearance*: The opposite of discovery denotes the capability of a system to detect when devices or services leave the network and to react accordingly.
- *Location Awareness*: Some applications require location-aware devices and services. Thus application developers should be able to *a)* find out the location of specific devices and *b)* find devices with respect to a given location in order to use services of these particular devices.

2.5. Technical

This section describes technical requirements to a composition solution.

- *Interaction with Heterogeneous Services*: Interconnecting heterogeneous services and devices (e.g., DPWS, REST, non-IP based) is necessary to develop smart home applications. Services might both reside on devices in the home or in the Internet (e.g., higher valued services like include weather forecast in heating control).
- *Extensibility*: Using new functionalities which are not foreseen at design time requires extensible tools and methods for application development. As an example, device discovery might be included later on but not in the first revision of the solution.
- *Data Manipulation*: Interacting with devices and services from different vendors requires mapping data representations and data formats and allows dealing with concepts that are modeled differently in different domains. This requires transforming an instance of a model into another model or transforming instances of two different models into a higher-level model.
- *Traceability*: Tracing actions (e.g., start of a process, invoking an event) is often required either for statistics or liability issues.

2.6. Security, Safety and Privacy

- *Process Safety*: Unsafe applications negatively impact devices or the environment in a way which is not foreseen by the developer and must be predicted to ensure process safety.
- *Confidentiality*: Information of the system should not be visible to anyone except for a defined group of people.
- *Authentication and Authorization*: Enabling confidentiality requires fine grained authentication and authorization mechanisms to access processes, devices and services.

2.7. Miscellaneous

This category contains all requirements that do not match the other categories.

- *Process Integrity*: Concurrent smart home applications should not contradict with each other (e.g., reduce and increase heating setting). Such contradictions can be detected during design or run time. During design time, the application developer can react accordingly. Capturing contradictions during run time requires the application developer to specify a

specific decision in advance (e.g., prioritize one application, throw an error, or make a compromise).

- *Transaction*: Executing a group of actions with transactional behavior maintains integrity of the application. However, in contrast to IT systems, rolling back actions within transactions is sometimes not possible. The actions may affect the physical world, which sometimes does not foresee being reverted (e.g., a sprinkler system).
- *Resource management*: Often cooperation with resource-constrained devices relies on a trade-off between functionality and resource constraints. It should be possible to specify how the application reacts to changing resources (e.g., cache sensor values in case battery level decreases).
- *Streaming*: Video, audio and data streaming is required in multiple domains such as in *security* (e.g., transmitting a video stream of a surveillance camera) and in *health care* (e.g., remote patient monitoring).
- *Concurrency & Scalability*: The system must be able to execute two or more concurrent processes at the same time. The need for more processing power should not grow exponentially with the number of users, devices or processes. This is often reached by a well-defined distribution.

This section presents a logical separation of requirements, each of which can be fulfilled individually. Application scenarios always consist of a complex set of these requirements and categories. In these sets, individual requirements are often mutually dependent. For example, periodically measuring sensor readings may require synchronously measuring data with a maximum age and real-time access to an actuator activity based on the readings.

Requirements related to marketing (e.g., low cost, shape of devices) are out of scope of this analysis.

3. Related Work

Currently no extensive list of requirements as presented in section 2 exists. Standardization organizations such as IETF and OASIS are focusing on providing interoperable protocols and do not explicitly address all requirements for composition solutions

The scope of existing standardization covers both applicability of emerging low power wireless technologies (e.g., 6LoWPAN[11] and IEEE 802.15.4) and usage of existing and mature protocols (e.g., IPv6[6]). Within IETF, the Smart Power Directorate [1] aims at bridging the gap of missing communication capabilities between the home automation environment and the backbone network by using IP as the common lowest layer for connected devices and Internet services.

Both IETF and mainly for security issues NIST (National Institute of Standards and Technology) are working towards interoperable communication protocols and their homogenous deployment and application in order to ensure seamless connectivity of all communication partners. Even industrial consortia are developing smart grid communication protocols such as the ZigBee Smart Energy 2.0 specification [2], EnOcean [3], and KNX [4].

Assuming protocol interoperability, Peltz [14] and Bohn [4] cover the required orchestration and choreography capabilities of SOAP Web Services. Peltz mainly focuses on Web Service Choreography Interface (WS-CI) [2], Web Services Business Process Execution Language (WS-BPEL) [1] and Business Process Modeling Language (BPML) [15] in enterprise infrastructures. Peltz does not address device-centric scenarios and thus does not cover the requirements for such applications. Bohn concentrates on the Devices Profile for Web Services (DPWS) [5] and the extension of WS-BPEL [1] to compose DPWS-enabled devices. Both approaches are only applicable in SOAP Web Services enabled environments. Technologies that are not based on SOAP cannot be integrated without complex gateways or proxies. In addition, both require programming knowledge to compose applications.

In [8], Dey et al. present *iCAP*, which is based on an end-user centric design and requires no programming knowledge to compose applications. Instead, *iCAP* expects the developer to express rules (e.g., *if <situation> then <action>*), define relation-based actions, and perform environment personalization. *iCAP* supports both real and simulated scenarios. However it does not explicitly support multiple communication technologies.

OSCAR [13] proposed by Newman, Elliott, and Smith also addresses device compositions for users with no deep technical knowledge. *OSCAR* allows to integrate arbitrary devices. It connects so-called *data consumers* with *data providers*, which is even feasibly for audio and video streaming. Unfortunately the paper does not describe technical details on how *OSCAR* provides interoperability.

VisualRDK (Visual Robotic Development Kit) [16] uses a visual language to rapidly prototype pervasive environments. However, *VisualRDK* does not support performing calculations or handling complex data structures.

In [10] Hague, Robinson and Blackwell present *Lingua Franca*, a composition system explicitly designed for home automation applications. *Lingua Franca* denotes a set of languages, each of which has its own focus and enables different users groups to create, modify and view applications. Some of these languages can be directly mapped to a base language, which is used by a runtime environment to execute applications. Vice versa, the base language can be mapped to some of the other languages. The paper describes the runtime environment but lacks details about its features and limitations (e.g., support for discovery).

[1] http://www.ietf.org/iesg/directorate/smart-power.html

[2] http://www.zigbee.org/

[3] http://www.enocean.com

[4] http://www.knx.de/

iCAP, *OSCAR*, *VisualRDK* and *Lingua Franca* all have in common that they address particular requirements. *iCAP* and *OSCAR* mainly focus on usability for home users. *Lingua Franca* also focuses on usability but provides different notations. *VisualRDK* provides a language to rapidly create applications. None of these approaches covers an extensive list of requirements as it is presented in this work.

To sum up, the lack of interoperable protocols and standards is in scope of protocol standardization organizations like IETF, OASIS and NIST and also industrial consortia like ZigBee, KNX and EnOcean. These efforts cover mainly the technical requirements presented in section 2. Further research approaches and also existing market solutions are investigating composition capabilities with particular emphasis on easy service composition even for end users[5]. However, all solutions are lacking crucial features. First, they do not inherently focus on both device-centric scenarios and higher-level Internet services at the same time. Second, they are lacking capabilities to meet a complex set of requirements as discussed in section 2.

4. Multi Layer Approach

This section proposes a multi layer architecture approach, which eases the design and implementation of smart home applications. It separates a platform architecture into multiple layers, each of which represents certain types of users. Employing this multi layer approach allows to group the requirements identified in section 2 and have them addressed by adequate stakeholders.

The multi layer approach distinguishes four different user types:

1. *Low-Level Developer*: The Low-Level Developer is a programmer who creates a software library. There is no need for domain-specific knowledge or knowledge about the platform as long as the library can be integrated in the platform. Such libraries can integrate Internet services, device communication technologies or provide a service or a functional block itself.

2. *Integrator*: The integrator integrates new features into the platform. These features then serve as a basis for smart home applications. An integrator must have some technical knowledge in programming and medium domain-specific knowledge. He should know which features can be integrated into the platform and how they should be designed to be useful for the application designer.

3. *Application Designer*: The designer of an application should have slight technical knowledge but comprehensive domain-specific knowledge. The designer should know the needs of end users, the technical possibilities as well as features of the platform to design smart home applications.

4. *End User*: In general an end user does not need to have any technical knowledge to run smart home applications. However, with some technical knowledge an end user is able to customize applications.

[5]http://pipes.yahoo.com/pipes/
http://www.fluxcorp.com/
http://www.kiwigrid.com/
http://www.ip-symcon.de/
http://www.loxone.com/

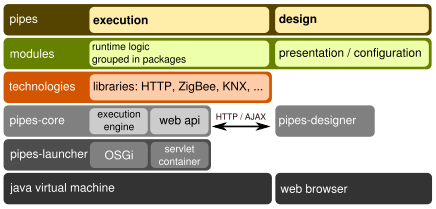

Figure 1. Architecture of WS4D-PipesBox

This role model and the multi layer approach architecture resembles concepts like programming in the large [7] or the process orientation in service oriented architectures [9].

The remainder of this section describes *WS4D-PipesBox*[6], a service creation environment (SCE) for composition of heterogeneous service oriented systems especially addressing Internet, enterprise and devices services based on the fundamentals in [3]. Afterwards it analyzes how the requirements for smart home applications are addressed in WS4D-PipesBox and thus how they are grouped to be fulfilled by a multi layer architecture.

4.1. WS4D-PipesBox

WS4D-PipesBox is software tool to mash up devices and Internet services to create new applications in service oriented device-centric environments. Overall it addresses the requirements with three key design principles. These principles are to (1) keep the underlying concepts simple and lightweight, (2) provide extensive extensibility and (3) highlight ease of use to address end-users with little technical knowledge.

4.2. Architecture

WS4D-PipesBox consists of several components as shown in figure 1. They can be divided into design and execution components. *pipes-designer* is an AJAX-based Web application to design smart home applications called *pipes* and runs in modern web browsers. *pipes-launcher* is a basic container running on a Java virtual machine that provides an OSGi environment and Java servlet engine to run and manage the *pipes-core* component and the *packages* that contain modules.

pipes-core is the central component of the *WS4D-PipesBox*. *pipes-core* provides a HTTP based web service API communication between the execution and design components. To execute pipes it contains an execution engine is described in more detail in the next section.

[6]c.f. Web Services for Devices Initiative, http://www.ws4d.org

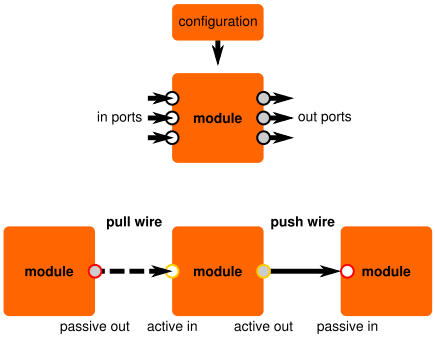

Figure 2. Modules, Configurations, Ports and Wires in the execution engine of PipesBox

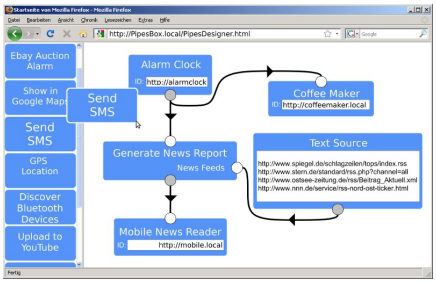

Figure 3. Mockup of the WS4D-PipesBox-Designer

Typically *WS4D-PipesBox* is running on a small embedded system that is located in the central control cabinet of a house where the home automation bus systems can be reached easily. It is connected to wired or wireless LAN where the enduser can configure and control the *WS4D-PipesBox* with a Web browser.

4.3. Execution Engine

The execution engine is based on the flow based programming principle defined by [12]. This principle uses software components with defined input and output ports for data processing. During the set up phase the process engine is responsible for instantiating components and wiring the input and output ports of instances according to a process configuration that was defined before. During execution phase the process engine is responsible for scheduling the instances depending on the data flow. This principle enables the easy creation of new process engines and is not limited to specific process execution languages.

As shown in 2, the pipes execution engine implements the flow based programming principles with a few extensions. In *WS4D-PipesBox* Components are called modules, have defined in and output ports and an a initial configuration. There are two kinds of wires to connect ports that implement a pull and a push behaviour.

4.4. Multi Layer Approach in PipesBox

WS4D-PipesBox was designed with the multi layer approach described before. The user types are mapped to the following layers in WS4D-PipesBox as shown in figure 1:

Technology Layer The *technology layer* corresponds to the Low-Level Developer. It consists of regular Java libraries called *technology* which integrate or implement a specific feature in Java. These libraries are not related to WS4D-PipesBox but can be developed independently. In the case of WS4D-PipesBox these libraries can integrate communication technologies (e.g., RS232, ZigBee, Bluetooth, UPnP or DPWS), data formats (e.g., XML, JSON or CSV), higher value Web services (e.g., Twitter or weather forecast), proprietary devices or software components (e.g., rule engines, encryption engines or data analysis engines).

To integrate a new technology in WS4D-PipesBox a Java library should be available which is in our opinion a low barrier to integrate new technologies.

Module Layer The *module layer* corresponds to the Integrator. It consists of *packages* that contain several *modules* to integrate a new technology. Packages are used to logically group modules which wrap the operations of an technology and homogenize technology-specific functionality for usage. A package is implemented as OSGi bundle that can be dynamically deployed in *WS4D-PipesBox*.

The task of the integrator is to wrap the operations of technologies in modules so that they can be used easily. A module has two representations. The design representation is used in the *pipes-designer* to build smart home applications using a graphical user interface. The execution representation contains the logic of the module that is used during the execution of the application.

The run-time logic must be written in Java. The *pipes-designer* representation currently must be written in Java Script but the Google Web toolkit will be used in future to write the designer representation in Java as well. Packages and modules provide meta data for the *pipes-designer* that are currently implemented with Java annotations.

Pipes Layer The *pipes layer* corresponds to the Application Designer who uses the *pipes-designer* in a Web browser to design new smart home applications called *pipes*. They consist of files containing the wiring between modules and the initial configuration.

As shown in figure 3 an Application Designer can create applications by combining modules. Modules have input and output ports. This ports can be either configured statically with an initial configuration or dynamically by using a wire. A wire connects an *outport* of one module with the *inport* of another module. Wires are based on weak typing and declare the interaction between modules during execution. Both module configurations and wire configurations describe a complete pipe and can be stored

Table 1. Requirement mapping

	PipesBox		User Types	
	Designer	Runtime	Integrator	Designer
Simplicity				
learning	+			
building & changing	+	+		
abstraction	+	+		
Modeling				
events	+	+		
expressive	○	○	○	○
statefulness			+	+
Timing				
hard-realtime	○	○	+	+
soft-realtime	○	○	+	+
data aging		+	+	
synchronity	+		+	+
periods	+		+	
Mobility				
discovery & disappearance			+	+
location awareness			+	+
Technical				
heterogeneity	+	+		
extensibility	+	+		
data manipulation	+	+	+	
traceability	+	+		
Security, Safety & Privacy				
safety	-		+	+
confidentiality	+	+		
authentication & authorization	+	+		
Miscellaneous				
integrity	-		+	+
transaction	-			
resource management			+	+
streaming			+	+
concurrency & scalability	+	○/+	+	+

in a file for exchange or deployed on *WS4D-PipesBox* for execution .

As wires and ports are based on weak typing, *WS4D-PipesBox* has no strict data model. The idea is to rely on primitive types like string, number, boolean and on formats like JSON or XML for complex types.

User Layer The top layer is the *user layer*. Users can exchange pipes and install pipes on their WS4D-PipesBox. A user can also install new packages and install, adapt or create new pipes. In the case of this paper pipes would represent smart home applications.

4.5. Requirement Analysis

Table 1 displays the result of the analysis how the multi layer architecture approach and WS4D-PipesBox address the requirements from section 2.

The column *PipesBox* indicates for each requirement whether it is directly addressed by WS4D-PipesBox or not (+: addressed, -: not addressed, ○: needs further research). Column *User Types* shows if a requirement must be addressed by a specific user type (+: has to address, ○: needs further research). The analysis considers only user types *Integrator* and *Application Designer*. This is because an *End User* does not need to consider any requirements. *Low-Level Developers* are not aware of the PipesBox framework when designing and developing the technology libraries.

The remainder of this section briefly discusses some of the requirements. WS4D-PipesBox addresses the *simplicity* category by design due to its graphical programming concept and the user type model. From the *modeling* category, *events* are supported by design. Modules in WS4D-PipesBox are triggered by incoming data, so events are supported in general. *States* of devices and applications are handled by integrators in modules and by designers in pipes, while *expressiveness* requires further investigations.

Requirements of the categories *timing capabilities* and *mobility* depend on technology-specific features. Hence for dedicated scenarios it is only possible to fulfill a given set of requirements by using specific low-level technologies which also fulfill the requirements. This must be developed independently outside the scope of WS4D-PipesBox. The basic *synchronization* mechanisms of a pipe are wires. More complex mechanisms can be implemented as *wrapping modules*. Simple *time-based triggers* are built in as modules but further mechanisms can be integrated as modules as well. *Real-time capabilities* are out of scope of current developments but focus of future research.

Technical requirements such as *heterogeneity* of smart homes are resolved by the high level of abstraction of the pipes-designer and the easy extensibility of the pipes execution engine with new packages. Some basic modules for *data manipulation* are available, further modules can be integrated. WS4D-Pipesbox traces the execution of every pipe that was instantiated with log files for *traceability*.

WS4D-PipesBox provides multi user support by maintaining user based sessions. These sessions can only be started or accessed after authenticating with typical form based login mechanism. When a session is started it will continue as long as there are running pipes related to. Users can only edit and control their own pipes so that basic *confidentiality* can be guaranteed.

WS4D-PipesBox can be used for both central or distributed composition of scenarios. Hence PipesBox has not necessarily a global view on processes and cannot guarantee *integrity* and *safety*. This must be addressed by the modules, pipes and the invoked services. As a process is described formally in PipesBox model checkers or other tools to check integrity and safety can be included. An alternative approach could be to use communities to rank pipes and thus create a moderate level of safety

Requirements of the type *miscellaneous* depend like the *timing* requirements mainly on the users which are re-

sponsible to design the applications accordingly. While *streaming* features require corresponding capabilities of the used underlying technology, *scalability* of WS4D-PipesBox mainly depends on the hardware platform and system specific characteristics. WS4D-PipesBox itself is not limited to the concept of modules and wires. However, further investigation is required on scalability requirements.

5. Conclusion and Future Work

This paper illustrates the potential of interconnected devices for multiple application domains in smart homes. Exploiting this potential requires individually composed solutions in contrast to off-the-shelf solutions or expensive installations performed by professional installers. The paper presents an extensive list of requirements as a guidance for developers of tools that aim at realizing smart home scenarios. Existing composition solutions only focus on individual requirements such as *usability* or *protocol interoperability*. A comprehensive solution, which provides means to cover all requirements, is still missing.

This paper proposes to satisfy the requirements on different levels by separating the architecture in multiple layers, each of which mapped to a certain user type and stakeholder. It further presents the *WS4D-PipesBox* framework that is based on this multi layer architecture. The analysis shows that the design of the *WS4D-PipesBox* composition framework successfully covers most requirements while still retaining end-user usability by hiding technology details. Only few requirements, e.g. *timing related capabilities*, cannot be realized either due to the dependency on Java and OSGi or because verification requires further research efforts. Other requirements such as *process safety* are not part of the native *WS4D-PipesBox* framework but can be implemented by certain user types due to the extensible nature of the framework. *WS4D-PipesBox* only depends on Java and OSGi for its runtime, other technologies can be integrated by specific libraries and extensions.

Future work will include missing requirements coverage with specific focus on the security, safety & privacy category. The challenge is to meet the corresponding requirements while staying agnostic of the underlying composition design. Also we will perform a usability study to proof and quantify what we claim about the *simplicity* of using WS4D-PipesBox (e.g., learning curve, ease of use). Future research further analyzes the question whether applications should be deployed as a centralized orchestration instance or in distributed component-based environments.

6. Acknowledgments

This work has been partly funded by German Federal Ministry of Education and Research (BMBF) under reference number 01S 08003I.

References

[1] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu. Web services business process execution language version 2.0, April 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

[2] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. T. Nagy, I. Trickovic, and S. Zimek. Web service choreography interface (wsci) 1.0. Technical report, 2002.

[3] A. Bobek. *Serviceorientierte Infrastruktur für vernetzte Dienste und eingebettete Geräte*. PhD thesis, Fakultät für Informatik und Elektrotechnik der Universität Rostock, Germany, Dezember 2008.

[4] H. Bohn. *Web Service Composition for Embedded Systems - A WS-BPEL Extension for DPWS*. PhD thesis, Fakultät für Informatik und Elektrotechnik der Universität Rostock, Germany, Februar 2009. ISBN: 978-3-86844-108-6.

[5] Dan Driscoll and Antoine Mensch. *Devices Profile for Web Services Version 1.1*. OASIS, 2009. `http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.html`.

[6] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871.

[7] F. DeRemer and H. Kron. Programming-in-the large versus programming-in-the-small. In *Proceedings of the international conference on Reliable software*, pages 114–121, New York, NY, USA, 1975. ACM.

[8] A. K. Dey, T. Sohn, S. Streng, and J. Kodama. icap: Interactive prototyping of context-aware applications. In *in Proceedings of Pervasive 2006*, pages 254–271, 2006.

[9] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[10] R. Hague, P. Robinson, and A. Blackwell. Towards ubiquitous end-user programming. In *In Proceedings of the 5th annual conference on Ubiquitous Computing (UbiComp)*, October 2003.

[11] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), Sept. 2007.

[12] J. P. Morrison. *Flow-Based Programming, 2nd Edition: A New Approach to Application Development*. CreateSpace, 2010.

[13] M. W. Newman, A. Elliott, and T. F. Smith. Providing an integrated user experience of networked media, devices, and services through end-user composition. In J. Indulska, D. J. Patterson, T. Rodden, and M. Ott, editors, *Pervasive*, volume 5013 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2008.

[14] C. Peltz. Web services orchestration and choreography. *Computer*, 36:46–52, 2003.

[15] R. K. Thiagarajan, A. K. Srivastava, A. K. Pujari, and V. K. Bulusu. Bpml: A process modeling language for dynamic business models. *Advanced Issues of E-Commerce and Web-Based Information Systems, International Workshop on*, 0:239, 2002.

[16] T. Weis, M. Knoll, A. Ulbrich, G. Muhl, and A. Brandle. Rapid prototyping for pervasive applications. *IEEE Pervasive Computing*, 6(2):76–84, 2007.