Service-Oriented Architecture for Smart Environments

Viktoriya Degeler*, Luis I. Lopera Gonzalez†, Mariano Leva‡, Paul Shrubsole§,
Silvia Bonomi‡, Oliver Amft†, and Alexander Lazovik*
*Distributed Systems, Johann Bernoulli Institute, University of Groningen, The Netherlands
Email: {v.degeler, a.lazovik}@rug.nl
†ACTLab, Signal Processing Systems, TU Eindhoven, The Netherlands
Email: {l.i.lopera.gonzalez, o.amft}@tue.nl
‡ Dipartimento di Ingegneria Informatica Automatica e Gestionale "Antonio Ruberti", "Sapienza", University of Rome
Email: {leva,bonomi}@dis.uniroma1.it
§Philips Research, Eindhoven, The Netherlands
Email: paul.shrubsole@philips.com

***Abstract*—The advances of pervasive technology offer new standards for user comfort by adding intelligence to ubiquitous home and office appliances. With intelligence being the core of some newly constructed buildings, it is important to design a scalable, robust, context-aware architecture, which not only has enough longevity and evolving capabilities to sustain itself over the building's lifetime, but also provides enough potential for additional features to be added to the core Building Management Systems (BMS). Such features may include energy preservation system, or activity-recognition techniques. Service-Oriented Architecture (SOA) principles provide great tools that can be applied to the smart buildings design, however certain specifics of pervasive systems should be taken into account, such as high heterogeneity of available devices and capabilities. In this paper we propose an architecture for smart pervasive applications, which is based on SOA principles and is specifically designed for long-term applicability, scalability, and evolution capabilities of a BMS. We validate our proposal by implementing a smart office on the premises of the Technical University of Eindhoven and showing that it complies with the requirements of scalability and robustness, at the same time being a viable BMS.**

I. Introduction

The advances of pervasive technology offer a new level of standards for user comfort within smart buildings. The tools to automatically adjust a building's behavior to satisfy the immediate user needs are becoming increasingly widespread, even if so far only on smaller scales of a single room or several interconnected devices. The growing trend of increase in numbers and popularization of smart buildings requires the standardization of smart systems, not only on the level of single applications, but also on the level of architecture requirements. There are several issues to be tackled on the way to full adoption of such systems. One of the problems arises from a large discrepancy between the average lifespan of a building, and the average useful lifespan of a smart system that is installed within one. The current smart houses, constructed with the use of the latest smart systems, will have those systems outdated after a decade, even though the building itself is still regarded as a newly constructed. This trend can easily be seen on the buildings constructed in late 90's - early 2000's, with smart lighting systems which today can not be easily overriden or changed to newly developed ones. Sometimes enormous amount of work is required to retrofit existing buildings with even a single newly developed system, for example, energy saving or smart heating mechanisms. By introducing SOA principles to buildings' design, the evolvability of buildings' smart systems will greatly increase, allowing older buildings to stay on par with newly constructed ones for much longer periods of time. Many current buildings come with pre-existing solutions that contain largely hardcoded behavior. "Turn off lights after several minutes of no motion" is arguably one of the most commonly used rules, defined on the lowest level of the lighting system. When combined with the absence of manual switches, the rule often leads to frustration among users due to a large number of false negatives. The deep hardcodedness of such a rule makes it practically impossible to use more sophisticated activity recognition techniques for presence monitoring. The additional complexity arised from inability to easily transfer the solution for one building to another one, which results in similar sets of simple rules being implemented in new buildings again and again, and prevents the system from growing in power and complexity.

A service-oriented approach to design and development of a building management system (BMS) that we propose allows to mitigate these issues. To validate our approach, we have implemented an intelligent office on the premises of the Technical University of Eindhoven, the Netherlands. Our architecture allows for addition of any number of complementary services, as long as they conform to the Software as a Service (SaaS) paradigm. Among modules added to our intelligent office are activity recognition, thermo-fluid dynamics, reasoning module that is based on constraint satisfaction techniques, etc. It is essential for longevity of a smart building system to change its behavior based on new user requirements without the need to change the basic installation and hardware. By introducing SOA principles we make it easier for the building to keep up with the evolution of smart systems. The building that hosts our intelligent office was constructed in seventies and required a big initial effort to add the capabilities for smart reasoning. However, the system behavior can now be changed effortlessly via the user defined rules, which users can add or remove dynamically. The additional reasoning allows not only to satisfy user requests, but also to choose the most energy

efficient way of doing it.

We describe the related research efforts in Section II. Section III overviews our solution, and provides details of the proposed architecture. Section IV describes the important services, and Section V provides the details of the installation and performance results. Section VI concludes the paper.

II. Related Work

Service-oriented composition techniques are sought to be applied to the smart environment systems middleware in many modern projects. The role of web services in the area of domotics is discussed in [1]. The authors identify the main scenarios of web services usage, discuss publish/subscribe standards for such services, and present a possible architecture of a domotic environment for an elderly person as a case study.

The importance of a common formal context model for a service-oriented middleware is discussed in [2]. Authors propose a middleware architecture which uses context representation based on Web Ontology Language (OWL). Similarly, in [3] authors tackle cross-domain data integration for energy smart platforms by linking building data in the cloud to create a graph of relevant information for building management using the Resource Description Framework (RDF) notation.

Diverse techniques are applied to the smart buildings projects' architecture design. The description of architecture patterns, common for such projects, is presented in [4]. This study shows the importance of layered structure of the architecture, and discusses the significance of every individual module within the full system. Thorough surveys of different intelligent building projects are presented in [5], [6]. The MavHome [7] project and ongoing CASAS [8] project aim at full building's automation by discovering patterns of devices' usage via appropriate learning algorithms. They provide predictions of future usage and produced many datasets of activities, sensor data, etc. iSpace and iDorm [9] project features a dormitory room which automatically adjusts to user's habits, minimizing explicit requests from their side. The SmartLab Research Laboratory [10] creates a model of context-aware environment, which may be used by other projects, built on top of it, such as Assistive Display, ubiClassRoom, Eldercare. The Smart Homes for All (SM4All) project [11] constructed a smart apartment in Rome, Italy, which features the usage of sophisticated AI planning techniques and the Brain-Computer Interface to simplify the home control for disabled people.

III. System Overview

GreenerBuildings (http://www.greenerbuildings.eu/) is a European FP7 project, which aims to create a smart automated environment that combines automation for user satisfaction with energy-efficient environment adaptation. As a part of the project, an intelligent office is constructed on the premises of the Technical University of Eindhoven, The Netherlands. The project allows its users (i.e. people within a building) to establish and modify the rules of the building's behavior, so that the system automatically adapts to their needs by using the context information. The project features advancements in many research areas, including wireless sensor networks, smart grids, activity recognition, thermo-fluid dynamics, etc. The GreenerBuildings project architecture is shown in Figure 1.

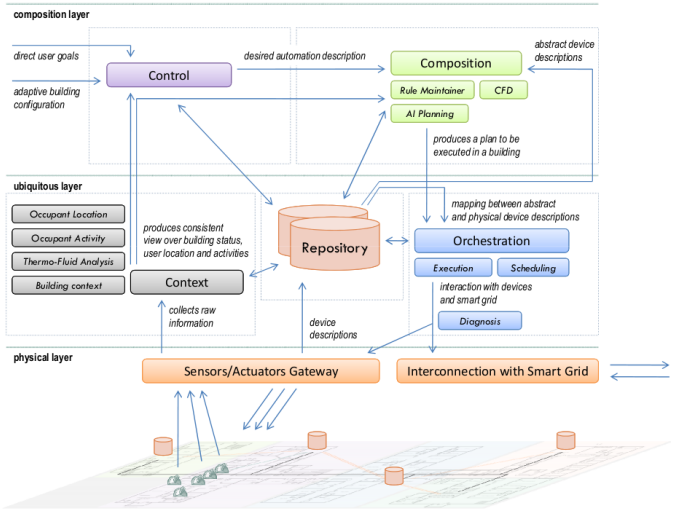

Fig. 1: GreenerBuildings Architecture

The Physical layer is responsible for handling the devices of the system, including sensors and actuators, and for the underlying low-level protocols. There are many different types of devices, among those are Plugwise devices, KNX controllers for blinds and heating system, motion, light, CO_2, and temperature sensors, etc. Though devices are operated via different protocols, the Sensors and Actuators Gateway service collects all information from raw devices and presents it in a uniform manner to higher levels of the system. The Interconnection with Smart Grid service provides the ability to be aware of the external energy pricing, and internal vs. external energy availability (e.g. from an internal wind turbine vs. external energy providers). The awareness about energy supply helps the GreenerBuildings system to adjust its demand and reduce the energy costs of the building operation.

The Ubiquitous layer ensures the proper operation of the whole system. The Repository is the database of the system. It contains all information about devices, their configuration, states, available actions that are represented as web services, and energy consumption. It also logs historical information about environment state which can be retrieved later for detailed analysis. The Context component collects information from sensors and transforms it into a high-level domain knowledge. This includes activity recognition, to represent such high-level activities as "a person is working with the computer", "there is a meeting/presentation in the room", etc. The Orchestration service is responsible for properly executing actions in a concurrent asynchronous way.

The Composition layer contains the Control service, which represents the system's interface to its users, including web interface, smartphones applications, dashboards, etc., and the Composition service, which is the main reasoning and decision making component of the system. The Composition, in turn, contains the Rule Maintenance Engine, which gathers information about user preferences, and decides, which goal state the system should be transformed into, to ensure the maximum user comfort and the minimum energy consumption; the Planner, which finds the actions to be executed to transform the system to this goal state, and the Computational

Fluid Dynamics (CFD), which handles the heating part of the building, including HVAC, air quality, etc.

IV. Application services

A. Sensors and Actuators Gateway

Current BMS have many sensors and actuators that often have to be designed specifically for the selected brand of the BMS, severely limiting the choice of devices. In other situations, when it is necessary to include sensors and actuators from different brands, arises a problem that different sensors have different communication protocols. The sensor and actuator gateway (SAGW) is a service that allows communication protocol standardization and interoperability. It translates protocol specific data to the GreenerBuildings communication standard and back. As new sensors, actuators and communication protocols become available, the SAGW can be easily upgraded to support them. The SAGW is not centralized, specific instances can deal with parts of the available network, instances take care of parts of the building, improving resource usage, e.g. bandwidth. SAGW is compatible with available networks in buildings making retrofitting the building much easier, and lowering the cost of installation even in new buildings as there is no need to add additional specialized hardware to achieve full building communication. The SAGW uses the distributed configuration service (DCS) to publish available variables, allowing context and orchestration services to communicate regardless of changing network configurations. This simplifies maintenance and makes the architecture robust.

B. Repository

The repository service manages the different storage needs. They are classified in three main categories: building description, service description, and context history.

Building description is a hierarchical tree structure that describes the building, and its physical locations. The repository uses the concept of a *cell*. It is defined as any physical location that has "contains" property, i.e. can contain other devices or other cells. It can also have any number of properties like location, area or volume. This representation supports even bigger abstractions like university campus or industrial parks, so several buildings can be managed together.

Service description contains all the information pertinent to a physical device or contextual variables. In the GreenerBuildings architecture, there is sensor and actuator data which corresponds to a physical device, and context data that is computed from the sensor data via activity recognition techniques. To simplify operation and component communication, all types of data are transformed into a single representation, *system variables*. It is a service that can be instantiated as many times as it is needed, with properties to support the differences between the types of data. The definition of the *system variable* uses two concepts: first it defines a service type, and second a service instance. The service type contains all the general properties that can be applied to a service instance. It is used to store things such as virtual representations, range of operation, control signals, expected values or ranges. The service instance is the implementation of the service type. It stores, among other things, the location where it is installed and the current value.

Context history stores the history of all variables. For each update, it stores the new value and the corresponding time stamp. This information can be used by reporting tools to generate building usage statistics. It is also used by the Context component to learn preferences and user behavior that allows to improve the comfort and energy saving.

C. Context

In most BMS control decisions are made over direct measurement of sensor values. This approach makes the control logic extremely complex. This causes building administrators' tendency to use rather simplistic interconnections between devices. The Context service is designed to simplify and enrich the available information about the environment, which can be used to make control decisions. It generates an abstraction layer that allows building managers to specify simple control rules over context variable instead of direct sensor input [12]. For example, the *presence* variable can be defined in terms of a motion sensor, a power consumption of the PC screen and a door sensor. As more sensors become available, more complex user activities can be recognized e.g. desk work, a meeting, or a presentation [13]. These high level constructs provide the required granularity to improve user comfort while maintaining energy saving [14]. Instead of only simple turn on/off functionality being available based on the presence sensor, with different activities being detected, more precise light scenarios could be applied, e.g dimming of overhead lights to a comfortable level.

The design of a perfect static system that provides comfortable room settings for all users under all conditions is not a feasible task due to large variance of user preferences and their perception of comfort. Therefore the context service learns from the user behavior and adjusts to better detect the context variables or give information to modify the control rules. Over time the system gets individualized to the preferences of the occupants of each space. The context service uses Context Recognition Network Toolbox (CRNT) [15] that allows to instantiate processing chains that produce the context variables required by the system. Each chain can be tuned to better take advantage of the available sensors that could aid to precisely compute the context variable. The chains use the DCS to locate the SAGW that provides the required sensor data, then publish the result to the event management system and finally send the value to the repository.

D. User Control

Many modern building automation installations are removing the ability for users to override default energy saving behaviours in order to provide a more quantifiable estimate on a return on investment of automation technology. For example, ASHRAE regulations in the U.S. are encouraging lighting control systems that are linked to occupancy sensors for energy saving, but as a result, there is a tendency to omit switches and dimmers. Moreover, facility managers are hesitant to promote the notion of user control in their infrastructure because they believe that users will select relatively high energy settings [16]. However, studies have shown that many users select energy settings that are significantly below recommended norms when they are provided with usable, personal controls, especially when the required environmental

conditions are task-dependent (such as computer work) [17]. Moreover, from a system view, the use of personal controls provides a rich source of contextual information about the satisfaction level of users in relation to the automatic behaviour of the system. To this end, the GreenerBuildings architecture specifies a user control component to support rich forms of interaction with the building system.

The Control service provides the UI between the user and the GB system in order to control its behaviour (both synchronously and asynchronously), based on the desired effects and possible contexts of an area within a building. It does this by enabling users (or building administrators) to design rules for energy efficient, comfortable building adaptation and by allowing users to stipulate (during actual usage) goals via an appropriate UI that can be translated into actuation commands that are compatible with the composition component. These goals are assumed to be contextually driven by occupant comfort needs and will therefore always override the default energy saving strategy of the rule maintenance system at a particular time. The control component also provides contextual feedback and feed-forward information that should be presented via events from the context and composition components.

The Control service translates UI events into control actions and aggregates incoming context data for rendering building-states to different users in the building. The translation can be made at multiple levels of abstraction from device-specific actions to topic-level actions (e.g. relating to desired light levels rather than to actual lighting actuator states). In order to translate user actions into energy-sensitive system goals, the control component receives events via Event Management service and binds them to user goal requests. Events are categorised by the device descriptions and are linked to a particular environmental effect within the building. They can be mapped to high or low-level building automation commands that are sent to the RME service. In addition, the control service provides timely feed-forward and feedback information about errors or conflicts that may arise when a user goal is requested. Here the rule maintainer forwards notifications and control possibilities to the control component when an actuator error is detected and the subscribed context variable can be used to give relevant information in the user interface. Finally, the control service is responsible for the development of building control rules and to stipulate building control goals. Here, the control component provides a simple means for a rule designer to develop new rules and delete old ones.

E. Rule Maintenance Engine

The Rule Maintenance Engine (RME) is a reasoning mechanism to find actions to be performed by actuators in response to environment events. It runs as a back-end service, with REST interface enabled to be used by UI to display information to users via Web or Android devices, or by other services to make modifications to the system programmatically.

The initial configuration of the RME system and the latest values of sensors are loaded at startup from the Repository, and initial environment check is immediately performed. This makes the system tolerant to failures and crashes, as it automatically returns to its latest state after restart. Users may override any part of the environment configuration or behavior rules dynamically, without the need to restart the system. Every behavior rule is a formula in predicate logic, it can be added in any form by the users, then it will be checked for correctness, consistency and it will be dynamically rechecked when new sensor readings arrive [18].

The RME UI dashboard contains the information about the current environment state; the commands, issued to the actuators, and whether they are executed or not; whether there are rules which cannot be satisfied at the moment; which manual goals were set by system's users previously, etc. This is one of the main dashboards available to users, through which they can control the system and keep track of its status.

The main source of events are sensors that detect changes in the environment. Since there may potentially be hundreds of events per second, the scalable and highly reliable event management system (EMS) is used to transfer this information. Effectively both low-level sensors and high-level recognized activities are represented uniformly within the RME, irrespective whether it is a single sensor which represents a pressure on a chair, or several sensors are combined together to represent a value of a domain-level "computer work" activity, which represents if a user is working with a computer at this time or not. The second way of obtaining events is the Control UI, where users may set their goals manually. Such user goals always override system's decisions. E.g., the rules may say that a temperature in a certain room may be as low as 19 degrees Celcius, but if a user specifies that she wants the temperature to be 22 degrees, the user command event will be generated and sent to the RME. When such an event arrives, the RME checks affected parts of the environment, and if any actions are required, they are sent to the Orchestrator service for execution.

F. Orchestration

The Orchestrator service is responsible for the execution of plans, generated by Composition services. Each orchestration represents a sequence of actions that need to be performed over the set of services that wrap physical devices. The actions are abstracted as web-services, and the orchestrator makes sure that they are correctly scheduled and executed at the proper time, with the proper parameters. If any contention problem is discovered, the plan is refused and a proper response is returned to the calling module. The orchestration service is synchronous, the plans are executed only if the devices are free, and in case of problems it re-executes the invocation. Each plan of actions is a tree of invocations, where the root and intermediate nodes are combined activities, and leaves represent atomic actions. Nodes can be nested in arbitrary depth level that does not need to be defined in advance. The orchestrator maps the abstract view in real time, with different threads and synchronisation points used to respect the different semantics of the activities. In smart environments devices can be used for different purposes by several actors at the same time. The concurrency manager module of the orchestrator is responsible for the execution of concurrent plans, guaranteeing that each resource is used at every moment by just one entity. In order to handle more execution in parallel, the orchestration engine is organized as a multi-thread server where each plan is managed by an execution process thread. The service also handles possible connection problems that can occur with SAGW, and notifies users if a service is unreachable.

G. Distributed Configuration and Monitoring Service

Distributed Configuration Service (DCS) is responsible for maintenance of information about the address, parameters, and (if applicable) physical location of every system service instance. Initially, when each instance starts as a service, it sends the update to DCS which stores the information about its location in a tree-like structure. Instances are grouped per components and location information is represented in form of endpoints (addresses) that point to specific instances.

DCS is represented as an Apache Zookeeper server that can have several replicas. Clients are the services that represent the instances of the system components. Each client service is connected to a single DCS server. Clients maintain a TCP connection through which they send requests to update the information about their physical location, get the responses which contain the physical location of other clients and send heartbeats (used for Monitoring Service). If the TCP connection to DCS server breaks, clients (system component instances) can connect to a different DCS server (replica), which contains the same information. This way fault tolerance is achieved and there is no single point of failure.

The system also provides Monitoring Service to represent the current state of each instance of system components and to have more information on its performance. Monitoring Service creates watchdogs for each service instance. As all its client components send the heartbeats, it is possible to keep the information in monitoring service refreshed.

H. Event Management service

The event management service (EMS) allows to filter events so that other services process only those events, which they are interested in. For example, in a sufficiently large building, there are several instances of the rule maintenance engine running, each responsible for a certain part of the environment. After initialization, the service contacts the event management system, and specifies the parameters of the events it is interested in, which include hierarchical physical location, type of services and devices, particular sub-system, such as lighting or heating, or the type of events. In the GreenerBuildings system the RabbitMQ messaging protocol is used as an underlying messaging system for the event management service. Due to the fact that events are handled in a unified manner, the event management service allows the system to be potentially extendable not only quantitatively, i.e. by adding more locations or devices, but also in a functional way, i.e. by adding new services to complement existing ones.

V. Evaluation

A. The smart office

To evaluate the architecture an intelligent office was created in the Potentiaal building in the Technical University of Eindhoven. Special care was taken to ensure that the office would allow to assess and test the benefits of the architecture in a real working environment. An office room and a meeting room where chosen for tests. Fine-grained activity monitoring and control are required in order to achieve energy saving[19]. The Potentiaal building was constructed in the seventies, therefore it had no centralized or automated building controls. To achieve the levels of detection and actuation a localized upgrade was done in the spaces chosen for the smart office. Lights where changed from simple on/off to dimmable, but the fixtures where kept intact. Also the coarse sectioning of the light installation was redone to have zones that could better react to the activities and had less lamps than the original sections. In the Meeting room electrical blinds where installed and the old radiator heating system was canceled for the room and replaced by two portable HVAC units. Energy harvesting wireless sensors and actuators were used whereever possible. This overhaul costed around ten thousand euros for parts and labor. In total the smart office keeps track of 113 sensor and context variables and controls 29 actuators. This increase on sensor density per room is necessary in order to achieve high levels of energy saving while maintaining user comfort.

The GreenerBuildings architectures is designed to run over tcp/ip protocol, this allows flexibility and easy deployment while keeping cost down by avoiding special needs in communication hardware. The smart office network exists as a VPN over the university network. Each room has a VPN enabled router that connects to the smart offices' VPN and at the same time provides a local network for the room. This eliminates the need of physically wiring a network between the two rooms and the machines running the other GB components.

B. Performance Metrics and results

There are two groups of metric used to measure performance of the architecture. The first group of metrics is aimed to evaluate the performance of individual components, giving an idea of scalability and robustness. The second group of metrics is aimed to evaluate the architecture as a whole, here the emphasis is in measuring the delay between detection to actuation. For the individual performance tests the components where evaluated as black boxes and simple machine resources utilization metrics where used to evaluate the performance: cpu, memory use, response time. The test methodology consisted of injecting progressively more simulated simultaneous requests until the amount reached the expected amount of requests from the entire Potentiaal building. For example Figure 2 shows the dependence of the CPU time on the number of simultaneous events.

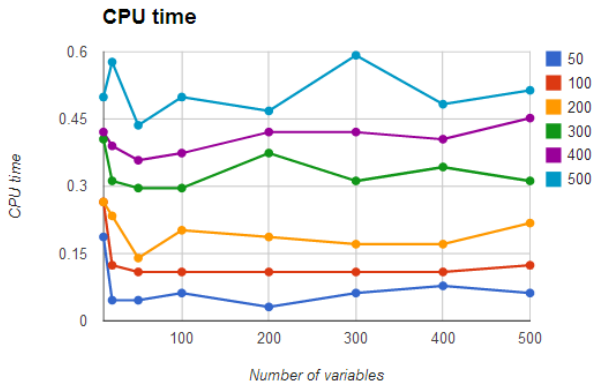

Fig. 2: RME CPU Time (seconds) per number of event calls.

The group test metrics focus on evaluating round trip timing and correlation. The round trip timing is measured as the

time it takes for the system to issue a command as a response to a change in an activity or sensor. The test was done for a general case scenario, where the components involved are: SAGW (sensor) , context, rule maintenance, orchestration and finally the SAGW (actuator). The selected device to measure was on of the infrared motion detector sensor associated to a desk presence activity. The desk presence triggers the turn on over head lamp rule, in proportion to the desk light sensor value. Finally the result is propagated to the orchestrator and to the SAGW that executes the action. The average round trip takes *0.34 seconds*, with a variance of *0.0142*.

The correlation metric determines how the system behaves when different actions affect the same sensor. This was tested on an office desk scenario, where the room lights and the desk overhead light affect the amount of light on the desk. The room light are set by a rule to be inversely proportional to the exterior light and the over head lamp are set by a second rule to provide enough light such that the desk light sensor reaches 700 lux. Figure 3 shows an example where the system compensates the effect of the room light with the overhead light. As the exterior light decreases the room light increases its value as a result the desk sensor stays constant. At on point the sensor detects the increase of light over the desk and the system immediately reacts by reducing the power of the overhead lamp. Finally when the influence of exterior light ceases completely the overhead light returns to full power.

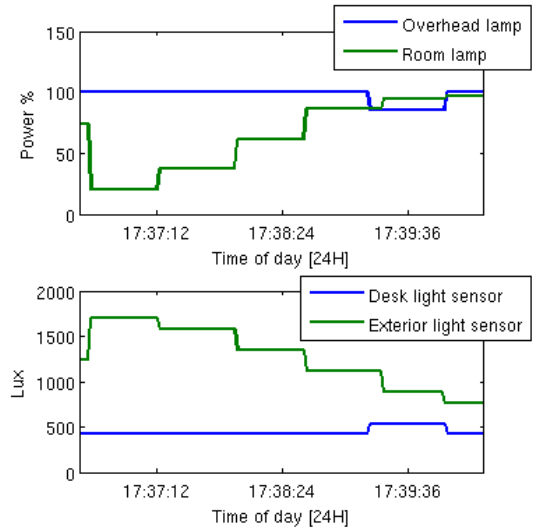

Fig. 3: Correlation test results. Room light compensates for the dwindling exterior light, and the overhead light reacts to influence on the desk light sensor of exterior and room lights

VI. Conclusion

Today we already have the technology to make our buildings smart. Now it is the priority to combine all the pieces, to create evolvable, portable, and easily adjustable middleware frameworks, which can be installed in new buildings and retrofitted to the old ones. In this paper we presented the architecture for smart buildings which is based on SOA principles. We showed, how different components may share information to achieve a common goal of increasing the comfort of a user. We showed, how the system, while becoming more automated, can still keep the user in charge of the environment. Our implementation of the smart office validated the responsiveness, scalability, and evolvability of the system, making it a viable choice for new BMS.

Acknowledgment

This work was kindly supported by the EU FP7 project GreenerBuildings, contract no. 258888.

References

[1] M. Aiello and S. Dustdar, "Are our homes ready for services? a domotic infrastructure based on the web service stack," *Pervasive and Mobile Computing*, vol. 4, no. 4, pp. 506–525, 2008.

[2] T. Gu, H. K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *Journal of Network and computer applications*, vol. 28, no. 1, pp. 1–18, 2005.

[3] E. Curry, J. O'Donnell, E. Corry, S. Hasan, M. Keane, and S. O'Riain, "Linking building data in the cloud: Integrating cross-domain building data using linked data," *Advanced Engineering Informatics*, 2012.

[4] V. Degeler and A. Lazovik, "Architecture Pattern for Context-Aware Smart Environments," in *Creating Personal, Social and Urban Awareness through Pervasive Computing*. IGI Global, 2013, pp. 108–130.

[5] D. Cook and S. Das, "How smart are our environments? An updated look at the state of the art," *Pervasive and Mobile Computing*, vol. 3, no. 2, pp. 53–73, 2007.

[6] T. A. Nguyen and M. Aiello, "Energy Intelligent Buildings based on User Activity: A Survey," *Energy and Buildings*, 2012.

[7] G. Youngblood, D. Cook, and L. Holder, "The MavHome Architecture," *Comp. Sci. and Eng. Dept. Univ. of Texas at Arlington, Tech. Rep.*, 2004.

[8] J. Kusznir and D. Cook, "Designing lightweight software architectures for smart environments," in *Intelligent Environments (IE), 2010 Sixth International Conference on*. IEEE, 2010, pp. 220–224.

[9] V. Callaghan, G. Clarke, M. Colley, H. Hagras, J. Chin, and F. Doctor, "Inhabited intelligent environments," *BT Technology Journal*, vol. 22, no. 3, pp. 233–247, 2004.

[10] D. López-de Ipiña, A. Almeida, U. Aguilera, I. Larizgoitia, X. Laiseca, P. Orduña, A. Barbier, and J. Vazquez, "Dynamic discovery and semantic reasoning for next generation intelligent environments," in *Int. Conf. on Intelligent Environments (IET)*, 2008, pp. 1–10.

[11] M. Aiello, F. Aloise, R. Baldoni, F. Cincotti, C. Guger, A. Lazovik, M. Mecella, P. Pucci, J. Rinsma, G. Santucci *et al.*, "Smart homes to improve the quality of life for all," in *Int. Conf. of Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2011, pp. 1777–1780.

[12] M. Milenkovic and O. Amft, "Recognizing energy-related activities using sensors commonly installed in office buildings," in *Int. Conf. Sustainable Energy Information Technology*. Elsevier, 2013, p. 669–677.

[13] P. Jaramillo and O. Amft, "Improving energy efficiency through activity-aware control of office appliances using proximity sensing - a real-life study," in *Int. Workshop on Smart Environments and Ambient Intelligence (SEnAml)*. IEEE, 2013.

[14] L. I. L. Gonzalez, M. Troost, and O. Amft, "Using a thermopile matrix sensor to recognize energy-related activities in offices," in *Sustainable Energy Information Technology, Int.Conf.* Elsevier, 2013, pp. 678–685.

[15] D. Bannach, P. Lukowicz, and O. Amft, "Rapid prototyping of activity recognition applications," *Pervasive Computing*, vol. 7, pp. 22–31, 2008.

[16] T. Moore, D. Carter, and A. Slater, "A field study of occupant controlled lighting in offices," *Lighting Research and Technology*, vol. 34, no. 3, pp. 191–202, 2002.

[17] A. Williams, B. Atkinson, K. Garbesi, E. Page, and F. Rubinstein, "Lighting controls in commercial buildings," *LEUKOS*, vol. 8, no. 3, pp. 161–180, 2012.

[18] V. Degeler and A. Lazovik, "Dynamic constraint reasoning in smart environments," in *IEEE Int.Conf. Tools wth Artificial Intelligence*, 2013.

[19] M. Milenkovic and O. Amft, "An opportunistic activity-sensing approach to save energy in office buildings," in *Int. Conf. on Future Energy Systems (eEnergy)*, ACM. ACM, 2013, p. 247–258.