Architecture pattern for context-aware smart environments

Viktoriya Degeler, Alexander Lazovik

Distributed Systems Group, University of Groningen, The Netherlands

{v.degeler,a.lazovik}@rug.nl

Introduction

The ability of pervasive systems to perceive the context of the surrounding environment and act accordingly proves to be an enormously powerful tool for raising immediate users' satisfaction and helping them to increase their own awareness and act in a more informed way. Therefore, recent years marked many smart environment solutions hitting the market and applying latest pervasive computing research advancements on an industrial scale.

Magnitude of context-aware smart spaces applications is enormous: it stretches from telephones that redirect the call to the room where the recipient is currently located, e.g. the Active Badge system (Want, Hopper, Falcão, & Gibbons, 1992), and simple coffee machines with the possibility to schedule the time of coffee preparation exactly to the time when you wake up to whole building automation systems with complex rules of behavior and planning techniques that are just waiting for your wink to launch the complex artificial intelligence reasoning that will understand and fulfill your unvoiced demands.

Going even further, smart environments matter not only on the Personal and the Social scale, but on the bigger Urban scale as well. Sometimes whole neighborhoods can be considered as smart spaces, as shown by many Smart Grid enabling projects (Georgievski, Degeler, Pagani, Nguyen, Lazovik, & Aiello, 2012), (Capodieci, Pagani, Cabri, & Aiello, 2011). By introducing small scale energy generating facilities, such as wind turbines or solar panels, it is possible for individual buildings to produce more energy than they consume at certain points of time. To avoid losing this precious energy (which becomes even more precious considering its “green” sustainable origin), peer-to-peer-like energy transfer connections are introduced between buildings, with full featured automated negotiation techniques that enable one building to sell excessive energy to another neighboring building. First field-testing projects, such as PowerMatching City project in the Netherlands (Bliek, et al., 2010), which features 25 interconnected households, show that not only such energy comes with a cheaper price, but also the “transfer overhead” is severely reduced, as now the average energy travel distance is much shorter.

As can be seen, context-aware smart environments come in many different faces and on many different scales, but underlying idea remains the same: the system is aware of its context, i.e. the environment around, is able to act accordingly in an intelligent, predefined, learned, or automatically inferred way, and is able to communicate to its users, thus increasing their comfort and awareness level as well. Seng Loke in his book (Loke, 2006) defines the three main elements of the context-aware pervasive system: sensing, thinking, acting.

In just a few years after the first introduction of smart environments, the topic became booming, and many projects both in research and in industry were dedicated specifically to advancements in this area. As happened in many other research fields where a big number of different research groups and industrial companies started to work separately on the same topic, in the context-aware environments area the problems that the groups face are to a large extent similar, and some of them were solved several times, sometimes in a similar manner.

One of such problems, and an important one, is the high-level architecture design of smart context-aware systems. Since the beginning of the 2000s, many projects have been designing and implementing smart environment systems from scratch. However, when looking post-factum at the architectures of these systems, one can notice a lot of similarities among them. With the same basic structure, the biggest differences usually arise at the level of individual components, aimed to satisfy different end-level requirements.

Naturally appeared the idea to unify the architecture design for such smart environments projects. Taking many successful and undergoing projects as case-studies, we tried to find the common structure, the common patterns, and in some sense the "best practices" that can help future projects to reduce the efforts spent on the general system frame, and redirect those efforts to more specific requirements that are unique in every project. The work of Preuveneers and Novais (Preuveneers & Novais, 2012) surveys similar efforts to find and study best practices on different levels of smart pervasive applications that were already done in previous studies, including requirements engineering, context modelling, development acceleration and code reuse. In this chapter, on the other hand, we focus on a pattern for architecture design of smart environment systems.

We will introduce several layers of the architecture that inevitably exist in one form or another, and discuss the possible components that may be parts of these layers. We will then discuss the common information flows within such architecture and mention the most notable problems, such as scalability and fault tolerance. Finally, we will present several case studies, successful or undergoing smart building projects, and show that the presented pattern can be easily mapped to their architectures.

Smart Environments

During the last years a lot of projects were dedicated to intelligent buildings automation. Though in this chapter we do not aim to provide an exhaustive review of such projects and for more thorough surveys the reader can refer to (Cook & Das, 2007) and (Nguyen & Aiello, 2012), the history of the most influential smart environment projects can be presented as follows.

It all started with the Active Badge (Want, Hopper, Falcão, & Gibbons, 1992) as early as in 1992. Though Active Badge most commonly cited as the beginning of the general context aware computing area, it can be seen that the main part of the project was concerned with making the environment (particularly, stationery phones) smarter. Thus Active Badge is as well the first project that was concerned with the smart environments, and implemented them.

One of the earliest projects aimed at full building automation was MavHome (Youngblood, Cook, & Holder, 2004) (MavHome, 2003), which started in 2000. The project was oriented at discovering patterns of device usage and occupants' behavior by utilizing different learning algorithms. The project produced a lot of datasets of activities, sensor data, etc., which were used to provide predictions on future usage. The conclusion of the MavHome project was also a starting point for the currently ongoing successful CASAS Smart Home project by the same university (CASAS, 2008).

iSpace, which started as iDorm in 2002 (Callaghan, Clarke, Colley, Hagras, Chin, & Doctor, 2004) (iDorm, 2002), project features a room in a dormitory of the University of Essex (United Kingdom) campus fully equipped with sensors and actuators. The project uses full range of devices, featuring temperature, humidity, and light sensors, door locks, infrared sensors, video cameras, as well as HVAC system, motorized blinds, window openers, and light dimmers. The system can remember user's habits and automatically adjust its behavior accordingly, so that explicit requests for actions from the user can be minimized, unless, of course, the user changes his or her habits.

The SmartLab Research Laboratory (López-de-Ipiña, et al., 2008) (SmartLab, 2006) was constructed in 2006 to create a model of interactions between people and the context aware environment

that surrounds them. This laboratory is used in several research projects, including Assistive Display, ubiClassRoom, and Eldercare.

The CASAS (Center for Advanced Studies in Adaptive Systems) Smart Home project (Kusznir & Cook, 2010) (CASAS, 2008) started in 2008 and has since produced a lot of publications both in academic press and in mass media. The smart environment for the project is a duplex apartment at the premises of the Washington State University. The apartment is equipped with a grid of sensors, including motion, temperature, power meter. The project heavily relies on Artificial Intelligence techniques such as Machine Learning in order to automatically recognize patterns of occupants behavior and automate the building to provide help and increase occupants' comfort.

As a part of Smart Homes for All (SM4All) project (Aiello, et al., 2011) (SM4All, 2008), which also started in 2008, a smart apartment was constructed in Rome, Italy. The project implemented sophisticated AI planning techniques, which produce a set of actions to adapt the house to user's needs in every possible situation. The breakthrough of the project was the application of the Brain-Computer Interface, a great help for many disabled people, which features the ability to read brain impulses of a smart home user and transform them into a certain desire about the smart home state, which in turn can be transformed into a set of actions for smart home actuators.

The e-Diana project (e-Diana, 2009) started in 2009 and was concerned with creation of a unified platform for all possible sub-systems of a smart building, such as security, lighting, power consumption, HVAC, etc. The project also aimed to improve energy consumption efficiency of such buildings and to provide better situation awareness for infrastructure owners.

The GreenerBuildings project (GreenerBuildings, 2013) started in 2010 and implements the intelligent office, constructed on premises of the Technical University of Eindhoven, The Netherlands. The project features the ability of users to modify the rules of office's behavior, which will then automatically adapt itself to their needs based on the context information. The project gives special attention to such issues of smart solutions as fault tolerance and scalability, which are essential for realization of smart solutions on a large scale, given hundreds of separate offices per building, or thousands of smart homes within a combined smart city.

2010 is also the start year of the ThinkHome project (Reinisch, Kofler, & Kastner, 2010), aimed at optimization of the energy efficiency while maintaining user comfort. The project uses knowledge ontologies for reasoning about the home states, and plans to provide a comprehensive knowledge base for evaluation of control strategies based on relevant building data.

There are also many specialized projects, for example EnPROVE (EnPROVE, 2013) or BeyWatch (Beywatch, 2008) that mostly deal with energy saving part of the smart environments, however, we mentioned here only some of the general broad-purpose context-aware smart environments.

We now will go into details and introduce components that are common for such projects.

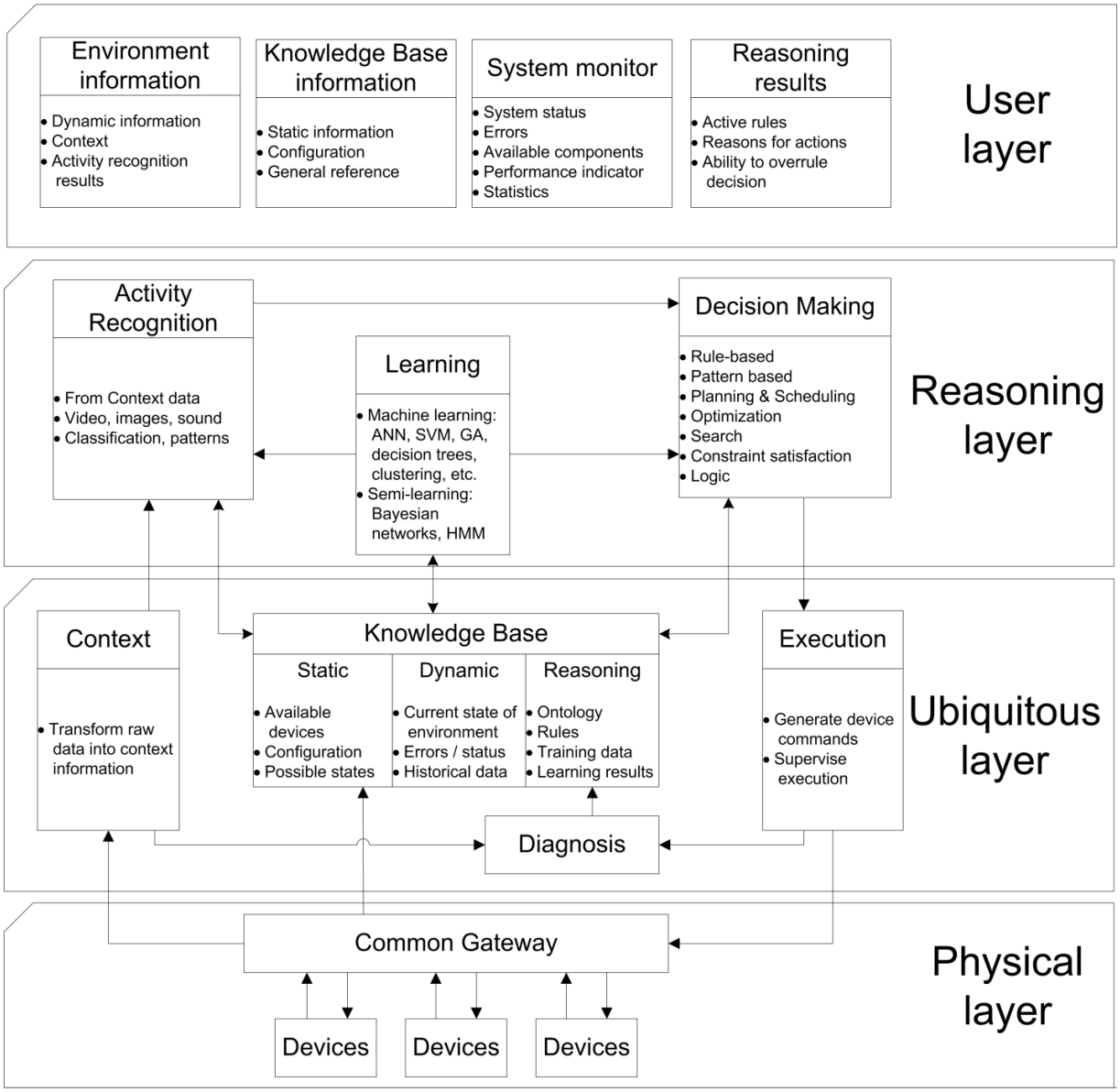
Architecture Overview

In this section we will present the design pattern of the smart environment architecture. The overview of the pattern is shown in Figure 1. We will show that the full architecture may be split into four layers, with several distinct components in every layer. Most component patterns arise from the architecture design similarities due to requirements that are common for all context-aware smart environments. It is important to note that components, which will be described below, are not exhaustive in terms of components' availability to the system. The components here are the backbone, but it is often the case that the actual implementation dictates for some support components, which either establish communication between other components, or act as watchdogs, proxies, monitors, etc. Also, if the system features a certain specific ability, such as a specific handling of heating mechanisms, or a special support for disabled users, more often than not this will require a separate component. Thus the system

that is described in this chapter should be viewed as extendable, with the ability to plug-in more components, if needed. And, to the opposite, some presented components and flows are sometimes simplified, combined, or removed altogether in projects of smaller scale. This possibility will be highlighted at the level of components.

The Physical layer contains all hardware parts of the system, which include all wired and wireless sensors, actuators, network layout, low-level protocols associated with them, etc. One of the main tasks of the Physical layer is to collect information about the environment and transfer it to higher layers. Low-level protocols may be implemented to provide a common gateway, which allows to unify interfaces, hide the specific hardware differences, and/or reduce bandwidth requirements by bundling the information. The second main task of the Physical layer is to invoke actuators in the environment based on commands, sent from higher layers.

Figure 1: Smart System Architecture Pattern



The Ubiquitous layer acts as an intermediary to the system components, and has several distinct responsibilities. First of all, the layer contains system's data storage, which means it collects and stores all the current and historical information about the environment, system configuration, system capabilities, user preferences, etc. The layer also should contain an information processing component, capable of detecting simple sensor errors or faults, transform information from low-level sensor values to high-level logical state of the environment, etc. On the actuation side the Ubiquitous layer is responsible for transforming commands from the Reasoning layer to low-level commands that the Physical layer is capable to execute, and making sure they are passed to the Physical layer in a concurrent non-blocking way.

The Reasoning layer is the layer where system's logic resides. It contains all components that are responsible for decisions on system's actions, be it a simple logic defined through strict if-then rules, or sophisticated AI techniques, such as planning or scheduling actions. The layer may also contain activity recognition or learning components, which should improve the automated system's response.

The User layer presents information about the system to its users. It contains two main parts. The first one presents information about the environment, current user preferences, reasons for certain decisions that the system makes, and allows a user to modify the configuration of the system according to her or his needs, enter new rules of execution, or overrule system's decisions. The second part provides meta-information about the system itself, such as the status of all components, whether they are working properly, statistics, resources consumption, etc.

We will now describe each layer in details.

Physical Layer

The main part of the Physical layer is, as the name hints, physical, i.e. devices that are implanted in the environment. All groups that decide to implement a smart home environment face an unavoidable issue from the very beginning: the heterogeneity of the devices they plan to use. Even now, while some companies started to specialize on providing combined sets of sensors and actuators, there are still a lot of special devices tailored to a particular need with distinct and possibly proprietary interfaces and communication protocols.

Thus it is essential to unify the interface and data gathering before sending the data further into the system. Not only such unification follows the famed low-coupling architecture principle, and makes it easy to add, remove, or change devices both individually, and as a whole type, but it also keeps all other parts of the system device-insensitive, so in its pure form the change of a device will not require a single line of code to be changed anywhere past the Physical layer.

Thus the essential part of the Physical layer is the Gateway, the component that initially collects data from devices and applies low-level transformation to it in order to send it further into the system in a uniform way. Note that the Gateway is highly hardware dependent, and will usually require changes in case of any changes of device types.

Of course, conceptually some other parts can be also treated as physical devices, especially information-providing ones, such as person's agenda, a call event over VoIP, or some electronic message from outside the system. Often such events are initially processed and entered into the system from the “top”, i.e. from the User layer, or even via some other distinct entrance point, directly into the Ubiquitous or the Reasoning layer. But we argue that with a good level of abstraction, which a well-implemented Gateway provides, adding such events to the system from the “bottom”, i.e. from the Physical layer, is also a perfectly viable solution that serves well to the unification of the event processing and information flows. In this case such event generators are commonly viewed and regarded as virtual or logical sensors.

Ubiquitous Layer

The Ubiquitous layer is the backbone of the whole system, the main support of all other components. It also contains main channels of information flow and storage, and in some sense the layer connects and helps in the interpretation of two different worlds: the device-level Physical layer and the domain-abstracted Reasoning layer.

There are several main components of the layer, and it is almost inevitable for all of them to be present in every smart environment system in one way or another.

Knowledge Base

We start with the Knowledge Base component. The database of the system belongs to this component, and for some systems the component will also be synonymous to the database. However, there is much more to it, first and foremost with respect to the types of information it handles. There are at least three types of information that are usually stored in the Knowledge Base, Figure 1 summarizes them.

The first type is the static information about devices. This includes the types of devices the system has, their communication protocols, whether they are sensors or actuators, the structure of readings they provide or states they can be set into. For configurable devices it also contains the configuration information. Though the name “static” implies that the information does not change frequently, it is nevertheless possible that the information will change automatically in the course of system’s operation. For example, the SM4All smart home environment (Aiello, et al., 2011) provides the automatic device discovery feature.

The second type of the Knowledge Base information is the dynamic one, and this represents the information that changes with high frequency, for example the current state of the environment, devices, or executed commands. Many systems prefer to send this type of information directly to relevant components (for example in the Reasoning layer) instead of sending them to the Knowledge Base and letting the Knowledge Base handle further distribution. This makes sense, since direct communication is also the fastest, and time of reaction is of utmost importance for the intelligent environment. However, the need for historical data collection is almost always a requirement, whether it is to update the training of a learning mechanism, to diagnose errors, or to show the history to a user. This means that even if the direct link for dynamic information transfer is outside the Knowledge Base, there should be a duplicate link which sends the data also to the Knowledge Base for storage and further retrieval and processing.

Finally, the last type of information in the Knowledge Base is almost exclusively used by the Reasoning layer, as it contains all the information, required for high-level reasoning. The exact model of information here depends heavily on what kind of reasoning the system uses. For ontology-based systems such as ThinkHome (Reinisch, Kofler, & Kastner, 2010), this will be the ontology of the system and the environment. For rule-based behavior the reasoning rules will be stored. Training data and learning results will be present for all systems that use machine learning in one way or another.

Context

The next component of the Ubiquitous layer is the Context. The main task of the context is to transform low-level raw data gathered from devices into a higher-level information, usable by the Reasoning layer.

One type of such processing is data packaging. Some sensors, for instance an acoustic sensor, send information with a very high frequency. It may be the case that the higher level components do not need such detailed information. Some simplified systems may only need to know, if there is a sound, or not, or the volume of sound, thus a lot of data transfers may be avoided by combining the information on

the Context level and only sending the results higher into the system. Not only the bandwidth is saved, but also it removes the need for the Reasoning level to have a lower level representation of the device, and allows it to think in "domain-level" terms. Other examples include simple error filters that work nicely for such sensors as motion or light sensors, which for the most part send correct readings, but may occasionally send faulty ones. Such outliers are easily detectable by comparing them with neighboring readings.

It should be pointed out here that the Context component in its pure form does not involve any kind of domain level reasoning, such as activity recognition. The Context instead must prepare the data for the high-level reasoning by abstracting some devices and transforming the data from other devices. As an example, let us take the presence detection. Though in its basic form it is a simple mapping with the motion sensor, the recognition of presence in the room already reasons and operates in domain level terms. To increase the sophistication level, other sensors may be used in later versions of the system in order to get better recognition rate (such as RFIDs on entering people, video stream, etc.). This will change trivial mapping into intricate reasoning system. Thus from the beginning such reasoning should be placed into the Activity Recognition component of the Reasoning level.

Execution

The Execution component is in some sense the exact opposite to the Context. The task of the Execution is to transform action goals received from the Reasoning layer into executable actions that can be sent to devices. An important addition to the task is also to oversee the correct execution of the commands by devices.

It should be noted that the Execution in its pure form, as well as the Context, has absolutely no domain-level reasoning, i.e. it should not decide which command to execute out of several possibilities (any form of such reasoning belongs to the Reasoning layer). A good example is that the command to the Execution to "turn on a lamp" should also specify exactly, which lamp should be turned on in case there are several of them. If, on the other hand, the command is general, as in "turn on anything that provides light", the Execution then also assumes some responsibilities of the Reasoning layer components as there may be several ways to satisfy the request (e.g. turning any one out of several available lamps in a room), and the Execution component must be able to choose one of these several available executions by using some criteria. On practice it still may be a viable solution, in order to reduce the complexity or simplify the architecture, but the system architect in this case should always be aware of this mixing of responsibilities, understand the reasons for them, and evaluate alternatives.

Even with this being said, the Execution still has (and must have) some form of reasoning, on the level of particular devices. For example, it must be able to match the correct execution action with the desired end-state of the device. Also, if some command always involves actuation of several distinct devices in a uniform manner, such a command can be abstracted on the Reasoning layer to a single atomic action, and only inside the Execution component it will be transformed into a series of commands applicable to each device.

Diagnosis

Finally, the last component of the Ubiquitous layer is the Diagnosis component. This component is optional, i.e. some systems choose not to implement it explicitly, especially at the early stages of smart environment development.

The task of the component is to monitor readings from sensors and execution results, check the correctness of the devices, and detect any anomalies, if possible. For example, many battery-powered devices tend to send erratic data when the battery is low. This may cause large problems at the reasoning level, if not detected earlier.

The diagnosis may also have its counterpart at the reasoning level, which will use domain data together with the information from the Diagnosis to forbid the usage of faulty devices, until fixed, thus restricting the available domain.

Reasoning Layer

The Reasoning layer contains the domain-level logic of the system. This is the most diverse layer as well, as every smart environment project has its own ideas on how the environment should reason and make decisions about the actions it should perform.

Over the years of context-aware systems research many different ways to model the domain-level information were devised, some general, some more specific to a particular task that the system was designed to solve. Among the most known high-level context representations are Resource Description Framework (RDF) (Lassila & Swick, 1998), W4 (Who, What, Where, When) Context Model (Castelli, Rosi, Mamei, & Zambonelli, 2006), RDF-based Web Ontology Language (OWL) (Antoniou & Harmelen, 2009), and Context Modelling Language (CML) (Bettini, et al., 2010).

An extensive survey of different context representation models is presented in (Bettini, et al., 2010). The choice of the exact context representation model influences heavily the capabilities for system's learning, activity recognition, and decision making, thus it is among the most important choices to be done during the early design of the smart environment system.

We split this layer into three components, however, smart environment projects history shows that projects may have any combination of these components intertwined in many different ways.

Learning

The Learning component is responsible for automatic learning of the best possible decisions and actions based on input data, which can either be a real-time data, or previously gathered training data.

The Learning component has a bit special place among all other components of the system. On the one hand, this component is optional, i.e. it is possible to construct a smart environment system without any learning incorporated, for example if it is a rule-based system. On the other hand, if the component exists, it takes one of the most important central places in the system.

Machine learning methods are numerous: artificial neural networks, support vector machines, decision trees, genetic algorithms, reinforcement learning, different clustering techniques, etc. They all are applicable for usage in smart environment systems.

Of course, when we speak about the learning capabilities of the system, usually it implies that the system has the ability to re-learn and re-train automatically when initial data changes, e.g. a user develops a new habit. However, there is also another possibility, a “semi-learning” system, so to say. In such a system the Learning component is not an integral part of the day-to-day system operation. Instead, the learning is performed using a standalone learning module at the beginning on some initial existing data, and results are entered to the system as unalterable rules. They are often represented by Bayesian networks or hidden Markov models. In such cases the Learning component may often be omitted from the operational architecture, as it indeed is not involved in the operational flows. When the need arises to relearn or retrain the system due to considerable changes in the outside world, the standalone learning module may be launched again, and the new operational rules will be entered to the system to replace the obsolete ones.

Activity Recognition

The Activity Recognition component does exactly what the name suggests: it gets the information about the current state from the Context, and applies internal knowledge to classify and define more high-

level information about the environment. For example, while the Context may send a reading from a motion sensor that there is motion in the room, the Activity Recognition will recognize that it corresponds to someone's presence in the room. Given the stream of video from the Context, the Activity Recognition may define a whole set of the new domain-level information, such as whether a person is working with PC, thinking, eating, moving around, etc.

Theoretically this component is not obligatory, as it is possible to make decisions directly based on the information from the Context. However, without the Activity recognition the complexity of decisions is severely limited, as they lack a big part of high-level domain information.

The activity recognition may include sound, video, or image recognition. Often it uses results obtained from the Learning component in order to classify and recognize the activity. Sometimes activity recognition may contain stricter definitions of what a certain activity means (such as a certain state of sensors will correspond to a certain activity), in which case the recognition itself checks the correspondence of the definition to the current state of environment.

The results of the Activity Recognition component will go into the Decision Making component, where, combined with the information directly from the Context, they will depict the full knowledge about the current state, which in turn will be used to make decisions.

Decision Making

The Decision Making component is what turns the intelligent environment from a silent observant into a resolute actor: it decides, which actions should be performed in a given situation with a given knowledge.

As with the Activity Recognition and the Learning components, the Decision Making component comes in many different forms, at least as many as there are fields in artificial intelligence and systems automation research areas. Some usable techniques include optimization theory, planning and scheduling, constraints satisfaction, search techniques, logical reasoning, ontological reasoning, reasoning under uncertainty, and many more.

The important difference to note is that decision making may be split into two types: instant and continual. Instant decision does not mean instant execution. However, it means that the decision, once it is made and sent to the Execution component, cannot be revisited and changed. Instead, the new feedback from the environment (even if it is a feedback about errors in execution) goes to the "new cycle" of decision making, and requires new decisions to be made. The instant decision making is easier from architectural point of view, particularly it goes well with stateless components, because every new decision can be made independently from previous ones.

However, sometimes instant decisions are not enough. Continual decision making usually involves several steps of execution within one decision. It also involves remembering the decision and revisiting it after receiving new feedback, possibly alternating some steps. Unlike instant decisions, continual ones usually require stateful components, thus are more demanding with respect to fault tolerance and general architectural cleanness.

User Layer

Though many projects opt not to give particular attention to interfacing with users, instead specifying UI as a part of some other architecture layer or component, we argue that it deserves a separate dedicated layer in the architecture.

The User layer provides a view of the system to the user, and, which is even more important, it gives the ability to change and fine-tune the system, to debug errors, to override system's decisions and much more.

In this section we will specify different parts of the system that require separate monitoring and control mechanisms.

The first component of the layer is the environment information. This is a monitoring component, which receives its information from two sources: the Context and the Activity Recognition. First of all, the component provides an important hint to the user about the view of the environment within the system, as generally it may be different from the actual state of the environment. Causes of this may be numerous: an erroneous reading of the sensor, a mistake of the Activity Recognition, missing information due to hidden changes that are not detected, etc. If the view within the system differs from the actual environment state, the decision may be incorrect or not optimal as well. Thus it is important for a user to be able to see the view within the system in order to be able to compare it with the actual state.

There is another important benefit of the environment information component: the increased user's awareness. Many studies show (Weiss & Guinard, 2010) that just by increasing users' awareness about the amount of energy they consume at certain times and when using certain devices, it is possible to reduce the total energy consumption, because users are more likely to decrease their usage of heavy-consuming devices.

Second component of the User layer is the knowledge base information and update. The static information about devices, their configuration, possible actions, etc., is a great reference to a user about the capabilities of the system. Whether or not the component should provide the ability to update static information depends on the general architecture of the system, particularly on where the entry point of such information to the system is located. For example, if the system should be able to automatically detect and configure the device for work, it may be wiser to restrict the ability to tamper with the device parameters through the user layer. More often than not incorrect detection may highlight deeper problems or bugs with device detection, which should be fixed, instead of just concealed by the manual correction.

The next component of the layer is the reasoning and decision making results. This information helps to understand the origins of system's actions, thus cannot be underestimated. It will show the reasons, why a particular decision was made by the system. For example, if the system decides to perform a certain action, this component will highlight, which rules exactly were activated. It is important to note that this includes information from all components of the Reasoning layer: the Decision Making, the Learning, and even the Activity Recognition. There is, however, no duplication of information with the environment information component, as the meaning of the information in these two cases is completely different. The environment information component must show the results of the activity recognition, in order to show, how the system perceives the state of the environment. The reasoning results component, however, explains how and why the decision was made. Therefore it will show in details, why the recognition algorithm classified the original information into exactly this activity, and not some other one. This knowledge will help the user to tweak the recognition algorithms if needed.

Finally, the last component of the User layer is the system monitor. Contrary to all previous components, instead of showing the information about the domain and the environment, this component shows the information about the system itself: health status of all components, their performance indicator, any detected status changes and/or errors, etc. This also includes detected errors in devices, which may require user’s intervention in order to check if device is working properly or indeed needs to be changed or repaired.

Operational Flows

Intelligent building systems are reactive, i.e. their behavior is a direct consequence of the information they got from outside. There are three general operational flows within the system, and every flow corresponds to a single information entrance point.

Of course, in our description it is assumed that all components are present in the system, which is not true for the general case, as many components are optional. If some component is missing, then every piece of information that should pass through the component is passed as it is (so we may assume that the transformation is the identity), and the component generates no new information.

Environment-generated

This flow is the most common one, as it starts with any registered change in the environment, and partially even with every new sensor reading.

The sensor reading is generated on the Physical layer and is sent to the Common Gateway, where it is converted to a uniform format. From the Gateway the transformed reading goes to two places: to the Knowledge Base for storage and further retrieval as historical data, and to the Context for immediate processing.

In the Context the reading is assessed and transformed from a raw reading data into a higher-level state. It may be the case that the reading corresponds to no changes in a state, in which case, depending on the system, the flow may either stop here (if further components are only interested in changes), or go further as usual. Either a state or a raw reading data (depending on the system) is also sent to the Diagnosis component, where it is checked for correctness.

The state is further sent from the Context to the Reasoning layer, starting with the Activity Recognition component, where recognition is performed to generate domain level knowledge. Then it is sent to the Decision Making component, where it is combined with all other available information and the system decides, whether a certain action should be performed.

In case there is a need for a certain action, the action is sent from the Reasoning layer to the Execution, where it is transformed to a set of device-level commands. And finally, those commands are sent to the Common Gateway in order to be distributed between the corresponding devices. They are also sent to the Diagnosis component for further checks.

Of course, in parallel with the flow described above, the information is sent to the User layer to be displayed in a timely manner. As soon as the Knowledge Base receives the new state, it is reflected on the corresponding dashboard. The environment information dashboard shows the results of the Context and the Activity Recognition components, and the reasoning dashboard shows the decisions made.

User-generated

The alternative flow is the user-generated one. This flow starts when a user shows the desire to change something in the way the system currently operates. For example, a user may override a certain decision, or change the priority of rules, or manually change a state of the environment, in case it was recognized incorrectly, etc.

The flow starts from one of the informational components of the User layer. When a user enters the change, it is processed and is sent to the respective component. For a manual change of the environment it would be either the Context, or the Activity Recognition, for a rule change it will be the Reasoning component, for a decision override it will be either the Reasoning, or the Execution component, etc. From there the flow goes further normally.

System-generated

The first type of the system-generated flows concerns the normal system operation, for example, when executing scheduled events. In such a flow, on earlier stages a plan or a schedule has been generated that required certain actions to be performed in the future. In such a case the internal clock is

set, and when the time comes, the action is automatically launched. The event usually starts from the Reasoning component, and goes further to the Execution normally.

Another type of the system-generated flows concerns the re-learning and re-training mechanisms. Usually the Learning component is updated during the course of system's operation, in order to correspond to changing conditions and requirements. Updating after every state change may be too cumbersome, especially for computationally expensive machine learning methods. Thus, re-learning happens either at some intervals of time or when a certain condition is met (such as a threshold for amount of changes is achieved).

Additional Challenges

For an intelligent environment that features a single room or a few rooms with no more than a couple of dozens of devices, the already described architecture will normally satisfy all demands of the architects and users combined. However, when a system becomes larger and grows to include several floors, a whole apartment or office building, or even several houses, new issues emerge that may render the intelligent environment almost nonoperational until properly solved.

The scalability of the system is the first such issue. First of all, a single server's CPU or storage power will be quickly outgrown, thus for any more or less large system several servers is a requirement. Currently many efforts are spent in the area of database systems on development of distributed fault tolerant databases, such as Hadoop (White, 2012), MongoDB (Chodorow & Dirolf, 2010), Redis (Sanfilippo & Noordhuis, 2011), Cassandra (Lakshman & Malik, 2009), etc. Such databases make a good base for extendable intelligent environments, as they already solve distribution, data replication, fault tolerance, and availability problems out of the box. However, not only the Knowledge Base needs proper scalability. The amount of sensor data grows with the number of devices as well, and at some point concurrency, queue processing speed and bandwidth issues may stop the system from further expansion. Thus it is also important to use proper solutions not only for data storage, but also for high-volume fast data processing. Such solutions as Twitter Storm (Twitter Storm, 2013) or RabbitMQ (Samovskiy, 2008) provide reliable ways for sending and processing large streams of data.

The Reasoning layer, however, is the one that may suffer most from system's expansion. The reason is that most of the machine learning, search and reasoning algorithms within the layer may be computationally expensive, with at least exponential solving time. While the parallelization and distribution on several servers may partially alleviate the problem, sometimes more fundamental changes to the algorithm will be required. One of possible changes is the usage of approximate algorithms (for example, greedy algorithms, or genetic algorithms) instead of exact ones for the search optimization reasoning. Another possible change is the splitting of the system into several independent subsystems of smaller size, and applying the algorithms within subsystems. While with this approach some dependency between parts from different subsystems may be permanently lost, if the subsystems have only weak and not important dependencies between each other this may be a big improvement in terms of system's reaction time with only minor consequences in terms of the optimality of reasoning results.

Another direct consequence of scaling the system onto several distributed servers is the need to increase the fault tolerance level. If the system works only in one room and on one server, crashes and other unrecoverable faults are rare and restarting the system is an unpleasant, but fast procedure that has overall light consequences. However, when servers become numerous, the rate of errors and crashes increases as well. The system should be designed in such a way that any single error will cause only a minor outage. So, for example, the system should be fully operational on fifth floor of the building even if the server that manages the second floor crashes.

This may be achieved through addition of special system-level components, i.e. components that manage the system itself. Monitoring and configuration component may keep track of all running

instances of components and their servers, check their health status through heartbeats, and keep track of their configuration.

In case a component dies, the configuration component will automatically restart it either on the same server, or on a different one, and reconfigure other components so that now they contact a new instance. The configuration component may also perform load balancing and other utility tasks. As with databases and data streams, there are solutions that may come handy for such component implementation, such as Apache Zookeeper (Apache Zookeeper, 2010) or Doozer (Doozer, 2011).

Case Studies

Finally, in this section we want to showcase several smart environment projects as case studies and discuss, how their architecture maps to the general pattern, described in previous sections. Except for small differences, it can be seen that the general architectures of the presented projects have many things in common. These projects are chosen due to several factors. First of all, their focus is on creation of a fully featured general intelligent building, which influences all aspects of building's operations, as opposed to specifically targeted projects, such as those that aim to create a smart lighting system, or those that only target efficient system's infrastructure, etc. Secondly, all chosen projects have constructed, implemented and tested an actual real environment, thus the architectures of these projects have proved their feasibility and validity. And finally, they mostly feature clear distinction of architecture modules, as opposed to several smaller projects, where some modules can be seamlessly combined, or removed altogether, due to their reduced functionality.

Even though the presented pattern is the most commonly used one for smart buildings, sometimes specific requirements may induce other constraints on the project and its architecture. For example, an emerging view of smart home architectures is viewing smart building environments as multi-agent. Cook in (Cook, 2009) defines four different directions in multi-agent research of smart environments: (a) multi-intelligent software agents, (b) tracking multiple residents, (c) profiling multiple residents, (d) multi-agent negotiations. The first direction usually assumes viewing every module of the system as a separate agent, with communication protocols guiding interactions between them. Surprisingly, such a view of multi-agent architecture can be very well combined with the pattern, presented here. In fact, in the same work Cook uses the MavHome project, which is one of our case studies as well, to describe how the agents can be organized in a hierarchical layered configuration. Other research directions view as agents either different people (in which case the smart system itself remains unified, but has to incorporate additional intelligence for distinguishing people), or different devices. In the latter case, especially if devices are highly mobile and autonomous, thus may be viewed as a complete system by themselves, the proposed pattern may be inapplicable or sub-optimal, and other agent based architectures may be explored, for example as described in (Spanoudakis & Moraitis, 2006).

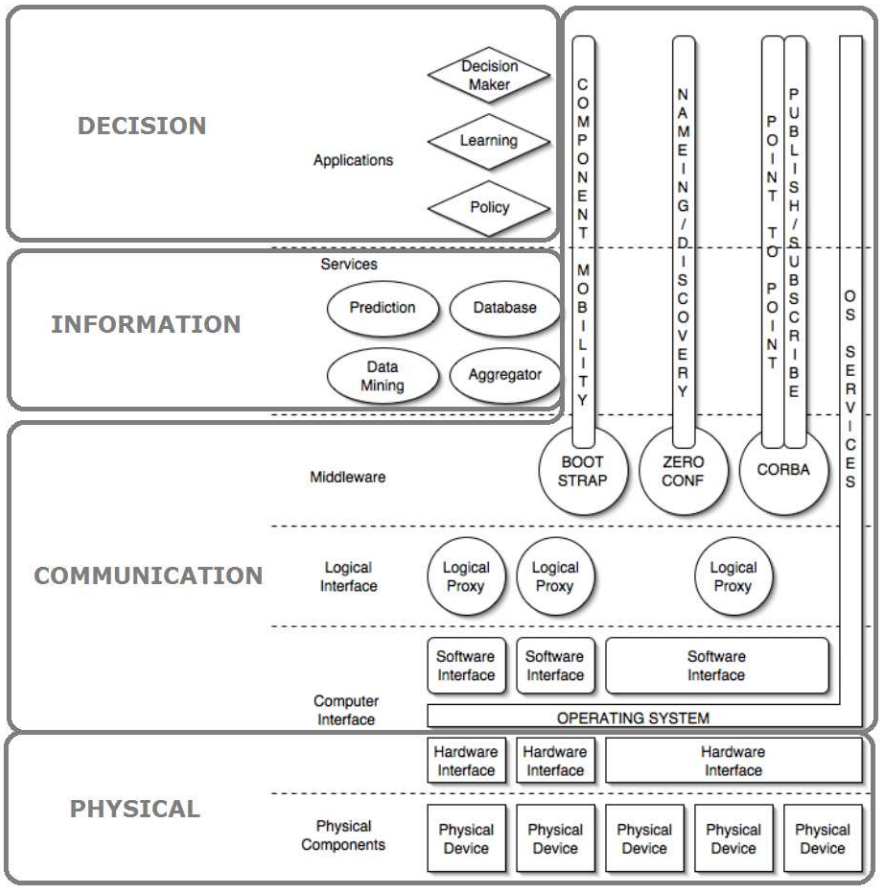
To avoid confusion, when referring to layers of respective projects and the layers of the architecture pattern, which is described in this chapter, we will refer to the former as “the project architecture”, and to the latter as “the pattern architecture”.

MavHome

Managing An Intelligent Versatile Home (MavHome) (Das, Cook, Battacharya, Heierman III, & Lin, 2002) project was one of the first scientific projects to create a functioning smart environment. The home system in the project acted as a rational agent, whose goal was to maximize comfort of its users and minimize costs of operation. The project used learning and prediction techniques heavily, to predict mobility patterns of the inhabitants and adapt to them in a timely manner.

The architecture of the project as described in (Youngblood, Cook, & Holder, 2004) is shown in Figure 2. Here we will briefly compare it to the pattern in the Architecture section of this chapter.

Figure 2: MavHome Architecture. Image source: (Youngblood, Cook, & Holder, 2004)



The Physical layer of MavHome exactly maps to the Physical layer as described in the pattern: it contains devices and device interfaces to higher components, reminiscent of the Common Gateway.

The Communication layer contains a lot of utility components that help to make the system operational, such as device drivers, operating system, proxies, and middleware. When comparing to the pattern, the Execution component is a part of this layer of MavHome. As we mentioned at the beginning of the Architecture Overview section, the implementation details are very specific to every system, so we avoid to include support components into the pattern, however they may very well be present in the high-level architecture overviews of particular projects, as can be seen in the Middleware sub-layer of the MavHome project example, where they take an important place in the implementation. There is one thing to note, however, that all device and hardware related utility software, such as drivers, operating system, proxies, etc. may also be conceptually viewed as a part of the Physical layer of the pattern.

The Information layer of MavHome contains aggregator, prediction, data mining and database services. It can be seen that it combines into a single layer parts of both the Ubiquitous and the Reasoning layers of the pattern. Namely, the Knowledge Base and the Context from the Ubiquitous layer, and the Learning and the Activity Recognition from the Reasoning layer.

Finally, the Decision layer of the MavHome project corresponds to the Decision Making component of the pattern.

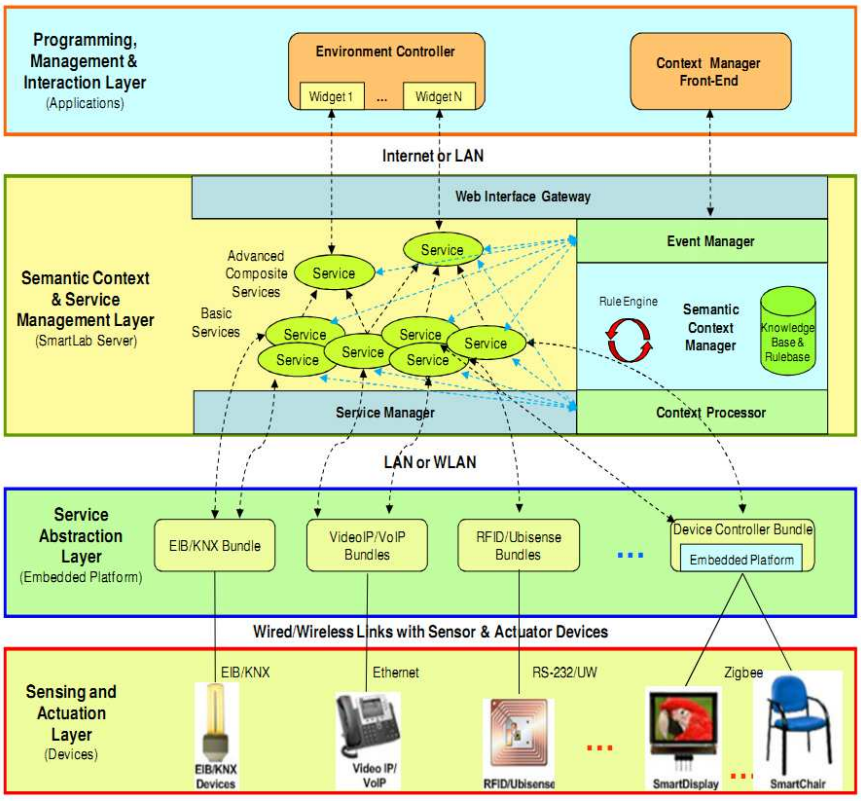
It should be noted that in the MavHome architecture there is no specific component or layer, responsible for interfacing with user, even though such interfaces (including mobile interface on PDA) actually exist. In case of their inclusion into the architecture picture, they may constitute the next layer, similar to the User layer of the pattern.

SmartLab

SmartLab is another project that has created a functioning smart environment (López-de-Ipiña, et al., 2008). The uniqueness of the project lies in the fact that the project itself features hardware and middleware parts of the environment (the Physical and the Ubiquitous layers in the pattern), with common interfaces for other projects to use and to create their own reasoning on top of it (the Reasoning layer of the pattern).The SmartLab environment was already used as a base for several other research projects, including Assistive Display, ubiClassRoom, and Eldercare.

The architecture of the project as described in (López-de-Ipiña, et al., 2008) is shown in Figure 3.

Figure 3: SmartLab Architecture. Image source: (López-de-Ipiña, et al., 2008)



The Sensing and Actuation layer contains all devices within the environment. They include EIB/KNX bus for lightning, HVAC, presence, temperature and motors on doors and windows, VoIP and VideoIP, Indoor Location System, etc. The next layer is the Service Abstraction layer, which transforms functionality of the devices from the first layer into software services. Together these two layers represent the Physical layer of the pattern, with the second layer representing the Common Gateway.

The Semantic Context & Service Management layer contains the Service Manager, which monitors the environment for activation and deactivation of devices thus for availability of services, the Semantic Context Manager, which stores knowledge about device in the common ontology, and the Web Gateway Module, which produces interfaces for third-party programs wishing to interact with the environment. This layer corresponds to the Ubiquitous layer of the pattern, with the Service Manager behaving as the Context component, the Semantic Context Manager behaving as static storage of the Knowledge Base component, and the Web Gateway behaving as the Execution component.

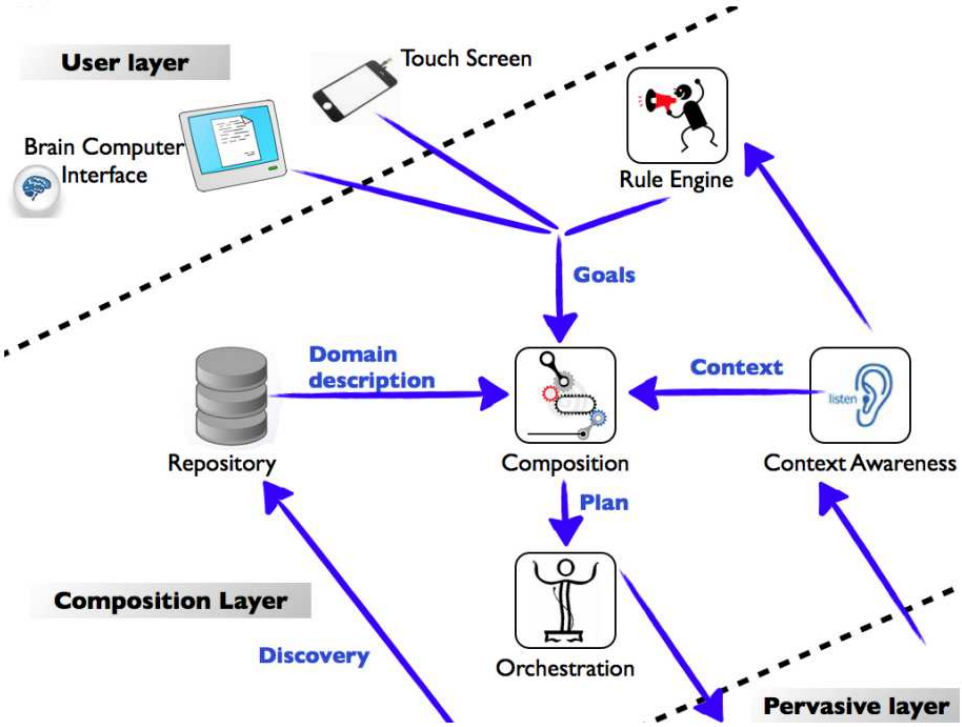
Finally, the Programming, Management and Interaction layer provides web-based interface for users of the SmartLab laboratory. The Environment Controller allows a user to manually operate the environment through a set of widgets, while the Context Manager Front-End offers a web interface for management of devices configuration, ontology, rule behavior, and tracking the system log and statistics. As can be seen, the layer closely resembles the User layer of the pattern.

Note that there is no layer similar to the Reasoning layer. As we already mentioned, the project provides capabilities for external programs to use the environment and middleware while applying their own reasoning. Thus such external programs will represent the Reasoning layer, when attached. Instead, the Semantic Context & Service Management layer provides all interfaces needed for external programs.

Smart Homes for All

Smart Homes for All (SM4All) (Aiello, et al., 2011) was a European-wide research project that had created a smart apartment in Rome, Italy. The project featured several innovative ideas within smart environments, including usage of the Brain Computer Interface for issuing the commands, using planning techniques for finding a set of actions for a complex commands, and sophisticated execution mechanisms to avoid concurrency issues when executing the commands.

Figure 4: SM4All Architecture. Image source: (Aiello, et al., 2011)



The architecture of the project as described in (Aiello, et al., 2011) can be seen in Figure 4.

There are three main layers. The Pervasive layer contains all devices and gives the possibility for devices to be added or removed dynamically through the usage of the common Universal Plug and Play (UPnP) protocol. As can be seen, the layer has the direct correspondence to the Physical layer of the pattern.

The Composition layer contains five major components. The Repository represents the Knowledge Base component of the pattern, and contains a database, which includes registry of current devices and their abstract types, description of available services, and information about the layout of a house. The Context Awareness collects sensed data and represents the logical image of the environment, thus being the Context component of the pattern. Though there is no specific Activity Recognition component from the pattern included in the SM4All architecture, some parts of it are also included in the Context Awareness. The Orchestration component controls the execution, i.e. it invokes the physical services and receives feedback about the status of invocations. As such it corresponds to the Execution component of the pattern. The Rule Engine component contains rules of the environment behavior and constantly checks, based on information from the Context Awareness component, whether those rules are satisfied; if so, it invokes the Composition component, which applies AI planning techniques to create a set of actions which are sent to the Orchestration. The Rule Engine and the Composition combined constitute the Decision Making component of the pattern.

The User layer provides access to the home system to its users. They may issue direct commands either through the touch interface or through the Brain Computer Interface. The User layer corresponds to the Reasoning results component of the User layer of the pattern.

GreenerBuildings

The GreenerBuildings project (GreenerBuildings, 2013) is the project that is dedicated to creation of smart offices in a green and energy-efficient way, while maintaining the high level of occupants' comfort. Occupants' behavior and activities are the key for adaptation to maximize the comfort, while choosing the most energy efficient state. The living lab setting is constructed on the premises of the Technical University of Eindhoven, the Netherlands. The project puts a lot of effort into the creation of a scalable, distributed, and fault tolerant solution.

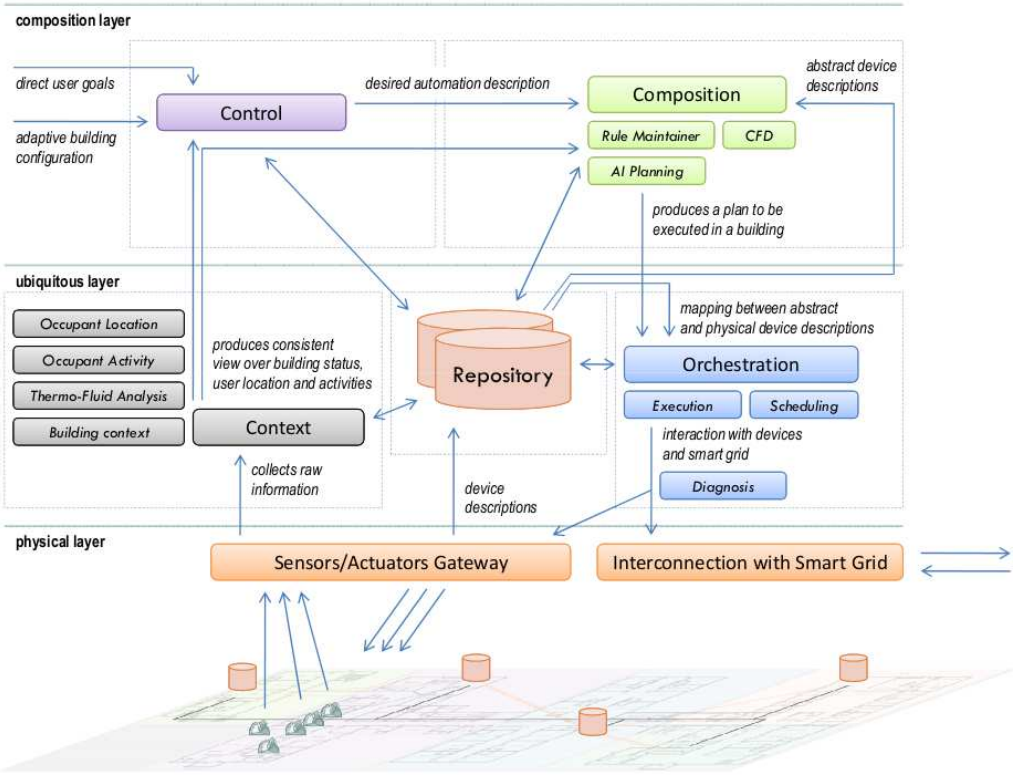
The architecture of the project is shown in Figure 5.

The Physical layer of the project contains all devices connected to the Sensors and Actuators Gateway, which sends the values further into the system. As such the Physical layer of the project resembles closely the Physical layer of the pattern. Note that the project layer contains one more component: the Interconnection with Smart Grid. Since the project puts a lot of effort in energy saving, the Smart Grid component provides the energy consumption and energy costs information. It also provides prices of energy from different energy providers, so that it is possible to choose the best price and the best time of task executions when the prices are the cheapest. The Interconnection with Smart Grid is the component, specific to the implementation of the GreenerBuildings, so there is no such component in the pattern. However, since it provides information, as other devices do, it can be viewed as a part of the usual Physical layer subsystem.

The Ubiquitous layer contains three main components: The Context, the Repository and the Orchestration, each having more subsystems within it. The Repository contains information about device types, device instances, and saves historical data for further retrieval. It corresponds to the Knowledge Base component of the pattern. The Context component collects information from sensors and transforms it into a consistent view of the environment. It also performs activity recognition, and as such it combines two components of the pattern: the Context and the Activity Recognition. The Orchestration performs

execution of commands and also diagnoses errors on the Physical layer. Therefore it combines the Execution and the Diagnosis components of the pattern.

Figure 5: GreenerBuildings Architecture. Image source: (GreenerBuildings, 2013)



The Composition layer contains two main components: the Control and the Composition component. The Composition component contains the reasoning of the system. The Rule Maintenance system within the component uses constraint satisfaction techniques to constantly check all rules that users have added to the system, and finds the state of the environment which satisfies all the rules. Planning component creates a set of actions to be executed by the Orchestration, and the CFD is the special system for optimal handling of the heating mechanisms and air quality within the rooms. Thus the Composition component is the Decision Making component of the pattern.

The Control component is the main system interface to a user. It shows system's parameters, and allows a user to issue direct commands or overrule decisions of the system. It also collects information about the users' satisfaction levels. As such it partially corresponds to the User layer of the pattern.

Conclusions

As we showed in this chapter, nowadays there are a lot of different initiatives which aim to create smart automated environments. For many of them, the architecture of the system is the first challenge they face, and as we showed, independently constructed architectures of many project still share similar component ideas, as a result of the inevitable process of finding the best solution and the best system design.

In the chapter we collected the knowledge, created by those projects, and combined it in order to describe a common architecture pattern. We showed, how existing project implementations resemble the pattern. We hope that this pattern will be of further use and helpful for many researchers and architects of the intelligent buildings in the future.

References

Aiello, M., Aloise, F., Baldoni, R., Cincotti, F., Guger, C., Lazovik, A., et al. (2011). Smart homes to improve the quality of life for all. *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE* (pp. 1777-1780). IEEE.

Antoniou, G., & Harmelen, F. v. (2009). Web ontology language: Owl. In *Handbook on ontologies* (pp. 91-110). Springer.

Apache Zookeeper. (2010). Retrieved 2013, from Apache Zookeeper: http://zookeeper.apache.org/

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

Beywatch. (2008). Retrieved 2013, from Beywatch project: http://www.beywatch.eu/

Bliek, F., van den Noort, A., Roossien, B., Kamphuis, R., de Wit, J., van Der Velde, J., et al. (2010). PowerMatching City, a living lab smart grid demonstration. *Innovative Smart Grid Technologies Conference Europe (ISGT Europe)* (pp. 1-8). IEEE.

Callaghan, V., Clarke, G., Colley, M., Hagras, H., Chin, J. S., & Doctor, F. (2004). Inhabited intelligent environments. *BT Technology Journal* , 22(3), 233-247.

Capodieci, N., Pagani, A., Cabri, G., & Aiello, M. (2011). Smart meter aware domestic energy trading agents. *Proceedings of the 2011 workshop on E-energy market challenge* (pp. 1-10). ACM.

CASAS. (2008). Retrieved 2013, from CASAS project: http://ailab.wsu.edu/casas/

Castelli, G., Rosi, A., Mamei, M., & Zambonelli, F. (2006). The W4 Model and Infrastructure for Context-aware Browsing The World. *Proceedings of the 7th WOA Workshop (From Objects to Agents).*

Chodorow, K., & Dirolf, M. (2010). *MongoDB: the definitive guide.* O'Reilly Media.

Cook, D. J. (2009). Multi-agent smart environments. *Journal of Ambient Intelligence and Smart Environments* , 1(1), 51-55.

Cook, D. J., & Das, S. K. (2007). How smart are our environments? An updated look at the state of the art. *Pervasive and Mobile Computing* , 3(2), 53-73.

Das, S. K., Cook, D. J., Battacharya, A., Heierman III, E. O., & Lin, T.-Y. (2002). The role of prediction algorithms in the MavHome smart home architecture. *IEEE Wireless Communications* , 9(6), 77-84.

Doozer. (2011). Retrieved 2013, from Doozer: http://github.com/ha/doozer

e-Diana. (2009). Retrieved 2013, from e-Diana project: http://www.artemis-ediana.eu/

EnPROVE. (2013). Retrieved 2013, from EnPROVE project: http://www.enprove.eu/

Georgievski, I., Degeler, V., Pagani, G. A., Nguyen, T. A., Lazovik, A., & Aiello, M. (2012). Optimizing Energy Costs for Offices Connected to the Smart Grid. *IEEE Transactions on Smart Grid* , 3:2273-2285.

GreenerBuildings. (2013). Retrieved 2013, from GreenerBuildings project: http://www.greenerbuildings.eu/

iDorm. (2002). Retrieved 2013, from iDorm project: http://cswww.essex.ac.uk/iieg/idorm.htm

Kusznir, J., & Cook, D. J. (2010). Designing lightweight software architectures for smart environments. *6th International Conference on Intelligent Environments* (pp. 220-224). IEEE.

Lakshman, A., & Malik, P. (2009). Cassandra: A structured storage system on a P2P network. *Proceedings of the 21st annual symposium on Parallelism in algorithms and architectures* (p. 47). ACM.

Lassila, O., & Swick, R. R. (1998). *Resource description framework (RDF) model and syntax specification.* World Wide Web Consortium.

Loke, S. (2006). *Context-aware pervasive systems: architectures for a new breed of applications.* Auerbach Publications.

López-de-Ipiña, D., Almeida, A., Aguilera, U., Larizgoitia, I., Laiseca, X., Orduña, P., et al. (2008). Dynamic discovery and semantic reasoning for next generation intelligent environments. *4th International Conference on Intelligent Environments* (pp. 1-10). IET.

MavHome. (2003). Retrieved 2013, from MavHome project: http://ailab.wsu.edu/mavhome

Nguyen, T. A., & Aiello, M. (2012). Energy Intelligent Buildings based on User Activity: A Survey. *Energy and Buildings* .

Preuveneers, D., & Novais, P. (2012). A survey of software engineering best practices for the development of smart applications in Ambient Intelligence. *Journal of Ambient Intelligence and Smart Environments* , 4(3), 149-162.

Reinisch, C., Kofler, M. J., & Kastner, W. (2010). ThinkHome: A smart home as digital ecosystem. *4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)* (pp. 256-261). IEEE.

Samovskiy, D. (2008). *Introduction to AMQP Messaging with RabbitMQ.*

Sanfilippo, S., & Noordhuis, P. (2011). *Redis.* http://redis.io.

SM4All. (2008). Retrieved 2013, from SM4All project: http://sm4all-project.eu/

SmartLab. (2006). Retrieved 2013, from SmartLab research laboratory: http://www.smartlab.deusto.es/

Spanoudakis, N. I., & Moraitis, P. (2006). Agent based architecture in an ambient intelligence context. *Proceedings of the 4th European Workshop on Multi-Agent Systems (EUMAS'06)*, (pp. 1-12). Lisbon.

Twitter Storm. (2013). Retrieved 2013, from Twitter Storm: http://storm-project.net/

Want, R., Hopper, A., Falcão, V., & Gibbons, J. (1992). The active badge location system. *ACM Transactions on Information Systems (TOIS)* , 10(1), 91-102.

Weiss, M., & Guinard, D. (2010). Increasing energy awareness through web-enabled power outlets. *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia* (p. 20). ACM.

White, T. (2012). *Hadoop: The definitive guide.* O'Reilly Media.

Youngblood, G. M., Cook, D. J., & Holder, L. B. (2004). *The MavHome Architecture.* Arlington: Department of Computer Science and Engineering University of Texas at Arlington.