Interoperability for Web Services based Smart Home Control Systems

Hannu Järvinen and Petri Vuorimaa
Department of Media Technology, Aalto University School of Science, Espoo, Finland

Keywords: Ambient Intelligence, Building Automation, oBIX, REST, Rule Engine, Smart Home, Web Intelligence, Web of Things, Web Services.

Abstract: One of the problems in smart home systems today is the lack of interoperability on different levels. While applying closed, non-standard, and complex protocols can cause the problem on a lower level, the architecture and design of a common building control system can cause it on a higher level. We present a solution for enabling the interoperability on the higher level in building automation systems with XML based rules and a Web API. To ensure the interoperability, we define requirements for Web services based building automation control systems. A standard building automation guideline, oBIX, is used to provide interoperability on the low level, and adopted for the rule management and description on the high level. The resulting rule engine architecture and implementation are evaluated against the requirements. The solution provides interoperability using standard Web technologies and supports employing several control systems simultaneously.

1 INTRODUCTION

In the Ambient Intelligence (AmI) world, our living environments consist of distributed embedded devices cooperating with one another while the intelligence of the system resides hidden in the network. These environments comprise of different kind of technology required to properly function together towards common goals. To achieve true interoperability between these technologies, four layers of interoperability need to be addressed: protocol layer, logic layer, semantic layer, and intelligence layer. In this paper, we present a solution for enabling smart home interoperability on the logic layer, and for supporting it on the semantic layer.

Common approach in smart home research has been to connect devices to a central control system that functions as a home server. Such a system can offer remote access interface for the users to manage their devices. The system can also offer means for a user to program certain logic by defining rules between the device states. While these kind of systems can help people to better manage their homes, they share two common constraints. In many cases, they are using closed and non-standard protocols for the communication, which restricts interoperability with generic devices. They also handle their logic internally in the control system blocking third party solutions out and forcing users to only use what is in hands. Hence, the system can function well with the devices that are connected to it, but is unable to interact with other systems or consider a new type of logic when built in functionality is felt insufficient. These independent systems can easily manage their internal behavior but the inter-system communication and logic is still a challenge.

In this paper, we present a solution for enabling higher level interoperability in smart homes. After a short introduction to home automation, we first discuss related recent research on smart homes and rule based systems, and then, considering the domain specific needs, define the requirements for Web services based building automation control system with basic logic interoperability and support for semantic interoperability. Next, we describe our architecture, use case and implementation, and finally evaluate the results against the requirements. The main contributions of our work are the requirements for the Web services based building automation control systems, and the rule engine implementation with high-level oBIX XML abstraction for the rule definition and management through a RESTful Web API.

2 BACKGROUND

Intelligent buildings have a long history and first attempts to make our homes smarter were planned already in the 1940s (Mozer, 1998). After that, the industry and research have periodically become interested in the idea of smart homes, and more lately the concepts of ubiquitous computing, pervasive computing and physical browsing have hyped the discussion on ambient intelligence up again.

Recent advancements in communication technology have brought us with possibility to network our home and building automation systems. The Internet is considered ubiquitous in modern buildings and provides a natural way to connect these systems together and with the rest of the IT world, offering the basic network infrastructure on both local and global scales. While the technology has been ready for smart buildings for a long time, they have not yet become reality for the people. Main reason for that has been the long battle between the standards on both national and international levels (Felser and Sauter, 2002). Agreements are not done on neither level, and today we have a huge set of competing device protocols.

The diversity of the device protocols forced to develop alternative solutions for modern Building Automation Systems (BASs). To cope with numerous non-interoperable standards, developers implemented first protocol converters between different devices, and later integration platforms, which are able to communicate over a variety of different protocols, and thus function as central servers for the device communication. An integration platform is basically a combination of hardware and software that is able to convert messages from different formats to others and communicate over different physical layers using a number of protocols (Järvinen et al., 2011) (Valtchev and Frankov, 2002). It can offer a high level interface to support information exchange with enterprise applications. Usually this interface, and more and more often the backbone network of the integration platform, is based on the Internet Protocol (IP) (Maile et al., 2007). Indeed, IP based communication is already widely applied in home automation systems. This allows exploiting successful and well-known Web technologies and tools in this domain.

Integration platforms partly solve the interoperability problem bringing information and controls from the devices available to the BASs. They unify numerous different protocols and data representations under one format, hopefully an agreed standard. However, after the interconnectivity, more is needed to make the devices to actually interoperate with each other. In practice, the interoperability on this level is realized with some kind of logic that makes decisions and strategies to execute commands based on the available information. In this paper, we refer to any system dedicated on handling this kind of logic in a building as a *control system*. Such system perceives the relevant environment states through, for example, current device state values, device history data, weather information and forecasts, or alternation of electricity price depending on the time of the day. It can also use any other external data available or a prediction made based on any of the mentioned data. Internal logic of the control system then evaluates this information with some reasoning and can decide to take actions based on the results.

In literature, various types of control systems have been studied. In addition to simple rule definition, other approaches such as fuzzy rules (Rutishauser et al., 2005) and Artificial Intelligence (AI) (Mozer, 1998) based solutions are suggested. What is common in traditional approaches is that they concentrate only on one type of control and handle it in a closed module with a specific user interface to manage the functioning.

The success of Web services as a solution for the integration of distributed heterogeneous systems has capitalized the research of service-oriented communications in the last decade. Web services are nowadays a *de facto* standard used to solve interoperability problems in distributed computing, which has motivated us to support interoperability offering a Web API for the rule management. Thus, in addition to devices, also the logic is controllable through the Web services interface.

In this paper, we concentrate on rule based control systems. However, we suggest supporting a whole horizontal layer of control systems simultaneously, thus enabling the interoperability also between the control systems. The focus of our research is on XML based rules in a distributed environment. In the next section, we describe earlier research on that area.

3 RELATED WORK

Rule based control systems generally describe rules with events, conditions, and actions. Such systems are able to automatically perform actions based on the occurring events and evaluated conditions. The idea is to have central management of the application logic, instead of implementing the functioning in separate programs, and to describe the rules with high-level, declarative syntax (Papamarkos et al., 2003). The composed Event-Condition-Action (ECA) rule structure has its roots in active databases. Originally, ECA rules were developed in eighties as a solution for the inability of the database management systems to trigger actions based on the alarms. Later on, ECA rules have been adopted by various other domains including XML, semantic web, and ubiquitous computing (Chakravarthy and Adaikkalavan, 2007).

Considering the event part particularly in XML documents, W3C has standardized Document Object Model (DOM) Event Module (Schepers et al., 2011) for that purpose. In addition to UI events, it defines mutation events which are triggered on any change to the structure of a document. Corresponding event handlers can thus be created in a standard manner to handle required actions when the DOM is altered.

Similarly for the condition and action part, the Extensible Stylesheet Language Transformations (XSLT) (Kay, 2007) provides a standard way for applying rules to XML documents. It can be used to create an XML document based on two source documents. The source documents are an input XML data file, and an XSLT rule file describing which elements and attributes of the input XML are included and how the output file is then constructed. The application area of XSLT is well defined and narrow, thus offering a common tool for the application development. Additionally, related to our solution, extensions such as XSLT 2.0 Extensions for Saxon[1] and EXPath HTTP Client[2] offer support for making HTTP requests from the XSLT.

For the semantic layer, W3C has a recommendation for Rule Interchange Format (RIF). It has been developed for the exchange of rules between the rule languages, such as the oBIX based rule language we propose in this article. RIF thus maps different rule languages together using a common syntax and semantics. Our approach for the interoperability here however focuses on the logic layer providing an open way for managing the rules through a Web API. Together with addition of semantic layer, rule engine would benefit from RIF, as it would be interoperable also with other RIF supported rule engines.

One common format for describing rules in XML format is the Rule Markup Language (RuleML). (Boley et al., 2001) It defines a general and flexible rule language and is specified by a group of international experts on the field. However, in this paper we wanted to concentrate on the smart home control systems, and to unify the markup and the interface for both the devices and rules.

A number of approaches for applying ECA rules on XML has been proposed. Typically, they consider XML as a data description format for the database, otherwise keeping a close relation to the traditional database systems. This means that the rules operate on XML documents but are themselves described in various non-XML formats (Bailey et al., 2002) (Papamarkos et al., 2003) (Bonifati et al., 2002), and that the operations are done with a local database.

[1]http://www.fgeorges.org/xslt/saxon-ext/

[2]http://www.expath.org/modules/http-client/

An approach from (Bonifati et al., 2001) defines the rules in XML, but also operates on XML documents locally. As the target documents are local, the system does not offer interoperability for distributed systems as we propose. Another approach from (Bernauer et al., 2004) tackled the problem of deciding on which events to react. As every modification of the document triggers Document Object Model (DOM) event, it is difficult to decide when conditions needs to be evaluated. Thus, they proposed a solution for detecting composite events for XML. However, they concentrate on the object node relations inside an XML document, and on the DOM events. This binds the solution to the local and complete XML documents. An article from (Papamarkos et al., 2004) presents an ECA language for Resource Description Framework (RDF). The language is able to function in distributed environments, as it supports path expressions with requests to resource URIs. The language itself is described in non-XML format, and the rule management is not discussed.

A recent research from (Leong et al., 2009) concentrated on rule based interaction. They employed ECA rules to enable interoperation between different heterogeneous subsystems in a smart home environment. They used SOAP as a backbone network protocol and stored rules in table form on a home application server. Subsystems reported their state changes to the home application server, which then fetched the related rules from the database, evaluated them and informed the subsystems if necessary. They defined an SQL based Application Programming Interface (API) for the management of the rules. The system could handle interoperation between the subsystems, but due to an additional API for the rule base, management of the rules could only be done using a specific user agent, not by other system modules.

4 REQUIREMENTS

Our approach is to gain the requirements for the Web services based building automation control systems from the properties of an Intelligent Agent (IA) defined in the Multi-Agent System (MAS) domain. An intelligent agent is a piece of software that is situated, autonomous, reactive, proactive, flexible, robust, and social (Padgham and Winikoff, 2004). We recognize four main properties of an IA that match with the properties needed for the control logic of a building automation system. These properties are: *autonomous* control, *reactive* responding to the changes in the environment, *proactive* observation of the environment, and *social* interaction with other agents.

We have derived the requirements from these properties and divided them into four categories. We suggest that the control descriptions should be based on standard XML description format, system be able to operate in distributed environments, support active knowledge base updates, and offer a standard interface for the management.

4.1 Description Format

To support social interaction, and for the simplicity of the control description, there are two requirements. The logic provided by the system should be described based on a common language utilized on the Web based building systems, similarly as all the data in the system should be presented using one standard XML based format. An XML based format is preferred due to its popularity in the Web, flexibility, both human and machine-readable nature, and direct support with Semantic Web techniques.

The description of the control logic in XML offers a way to hide the complexity behind a high-level abstraction layer. If a standard XML based building system language is utilized, there is no need to learn and implement new language syntax and semantics. Using a standard language also facilitates social interaction within different system modules and with the third party solutions. In addition to the XML encoding, the control logic should be described in a simple fashion. Declarative definition style defines the logic without describing the control flow. It contributes to the simplicity by offering a natural and intuitive way for presenting the logic. (Lloyd, 1994)

- R1: high-level standard XML abstraction for the control logic description
- R2: declarative control logic definition

4.2 Interaction in Distributed Environments

By their very nature, Web services based BASs are distributed device environments. In such environment, control systems should offer functionality for creating interoperability between various devices that provide their data and controls on the Web. Social interaction of the control logic and the devices can realize this behavior. Thus, control systems need to be able to both access web resources that are referred in their logic with URIs, and similarly to send requests to URIs for controlling web resources based on the following actions. A control system thus keeps up its knowledge base by perceiving the environment through references to multiple distributed web resources, instead of referring to path expressions within one XML document. This active inter-device communication also enables the interoperability in the action part of the logic. If the environment meets a certain state that changes the output of the logic, firing actions can reactively make requests to web based device resources, or similarly command other web resources, such as control systems.

- R3: ability to perceive the environment with web resource requests
- R4: ability to command actions with web resource requests

4.3 Knowledge Base Updates

Distributed nature and real-time requirements of building control systems require some considerations to be made for the knowledge base of the control system. One such requirement is the support for getting informed, or proactive/automatic updates of the web resource states that the system is aware of. A control system thus needs to be able to periodically request web resources for updating the knowledge base containing the relevant current perception of the environment.

- R5: active knowledge base updates

4.4 System Management

For the general interoperability, the control system management should be supported through a standard Web services interface. In addition, direct support for URIs as an addressing method simplifies the integration with the Web, and partly enables the social interaction. It also ensures that the logic is manageable the same way than other objects in the Web of Things (WoT) (Guinard and Trifa, 2009). While the control system expects devices to expose their functionality for the interaction, it should apply the same guideline to itself and provide its services through a Web API.

These requirements ensure that the application logic implementing the interoperation between the devices and subsystems is accessible for all system modules. Hence, the control modules of the system are manageable using the same standard interface than all the other modules of the system. This also allows logic to access other logic in the same control system, or in another type of control system.

Following these requirements in various control systems allows different modules to build a uniform system, where management of any module is accomplished following the same consistent principles. In practice, modules can exploit the functionalities offered by other modules, thus supporting interoperability on a higher level.

- R6: control system management through standard web interface
- R7: logic identifiable with URIs.

5 ARCHITECTURE

We have defined an architecture for smart home systems following the requirements from the previous section, and apply those across all the system modules. As depicted in the Figure 1, on the high level all modules of the system are using the same Web services interface for the communication between them. This allows flexible management of the modules to achieve a configuration of the system that the user desires. Opposed to more complex SOAP approach, we promote use of RESTful Web services to simplify the communication and the design, and to facilitate the development of the following implementations. It is notable, that the applied Web service technique has to be open and standard to achieve interoperability with third party solutions.

As the communication interfaces are unified into one standard RESTful Web API, it is easy for modules to communicate with each other. In our model, this is the only interface for the management of a certain module. This important design principle assures, that the management of the modules is open for the whole system. As seen in the Figure 2, this is a conceptually different approach than having, for example, the application logic managed inside a closed module that is configurable only by a specific user interface or application. Now, control logic can be easily created and modified using the Web API that the modules provide.

While usage of only one interface makes the system simple, it also sets specific requirements for it. The interface standard has to allow a flexible way for developers to define their own diverse data models for describing their data. It has to be flexible enough to also allow the management of the control systems and be able to describe their functionality sufficiently. The network includes various different types of modules, and there can be multiple instances of any module type functioning at the system simultaneously.

An integration platform brings the information from the home devices available to the system. It communicates using various underlying low-level device specific protocols and unifies their representations to a standard data format offering an interface

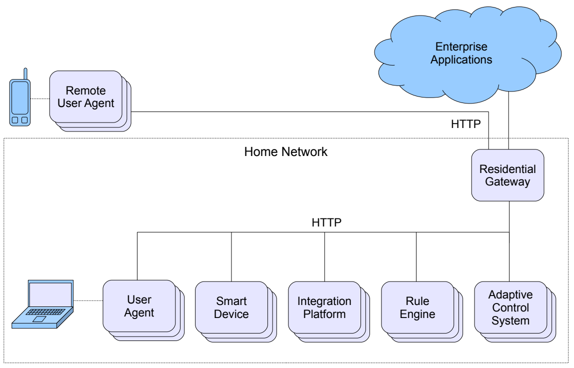

Figure 1: A smart home system model with distributed intelligence. A standard Web services based building automation guideline is applied for the communication.

for managing the devices that are connected to it. Smart devices instead are able to directly communicate using the standard backbone network protocol applied in the system. They do not need to be connected to any integration platform but if desired, they can do that as well. A smart device can also include application logic and it can, for example, directly communicate with other modules.

A rule engine is one type of control system. Using the standard RESTful Web API, other modules can create, modify and delete rules that run on the rule engine. Rules are a way to define interoperating logic to the system. Actions defined in an individual rule are executed if a certain logical condition is fulfilled. The rule engine then commands the related devices to take the appropriate actions. Rules are the most basic form of device independent interoperation in the smart home system. Similarly to a rule engine, adaptive control system is responsible for implementing interoperability to the system. The type of the logic, however, differs from the one in rule engines. The goal of the module is to learn from the behavior of the system and adapt the functioning to it. It can, for example, analyze the device history data and try to improve the efficiency and the user comfort. Because of this, the actual resulting behavior is not directly manageable but the related parameters are.

6 IMPLEMENTATION

The implementation focused mainly on the rule engine and the smart device but also included an integration platform and a couple of devices. As depicted in the Figure 3, the rule engine played a central part in the system. The following scenario was applied to test the implementation:

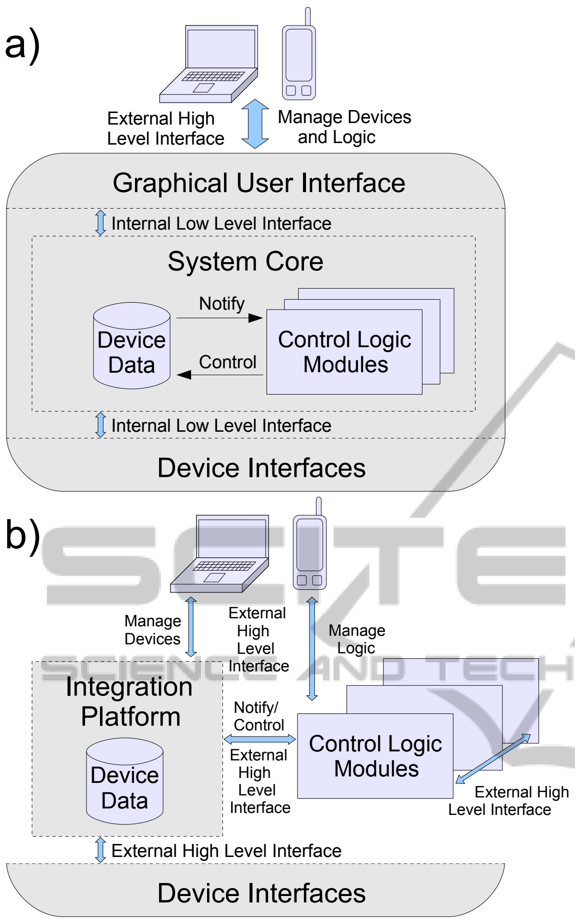

Figure 2: Communication between system modules in a system using a) Closed Control Logic; b) Open Control Logic.

Michael is watching television at the living room but wants to arrange some things at home at the same time. When he enters a room to pick up something, the light goes on in the room. When he leaves the room to put the thing in its correct place, the light switches off. However, when wandering around the flat he does not miss anything on the television, as always when he leaves the sofa, the program pauses. Similarly, when he sits down to the sofa again, the program continues.

For our implementation, we decided to use the open Building Information eXchange (oBIX) (Considine and Ehrlich, 2006) from the Organization for the Advancement of Structured Information Standards (OASIS). The data format is based on XML and defines, similarly to any common programming language, a small set of primitive data types for describing the data. It supports definition of contracts based on the primitives to allow developers to define data structures for their own devices and systems. The

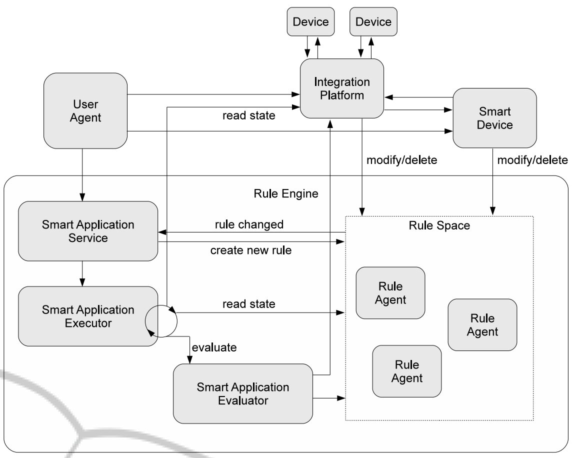

Figure 3: Communication flow of the implemented system.

extendibility suits well to our needs, as we want to describe multiple types of devices and control systems. The specification defines both SOAP and HTTP REST bindings. To simplify the development of the applications, we have selected to use the latter. An additional protocol layer is left out and the information can be directly read using an HTTP browser.

The environment had two rooms, both having three lights. For connecting devices to the integration platform, we decided to follow the same communication method than the rest of the system. Thus, the integration platform works as a standalone server and devices can connect to it through the Local Area Network (LAN). A camera based tracking system was used to get the location of the user. The tracking was implemented as a smart device and included additional logic. It used Linear Discriminant Analysis (LDA)[3] to predict if the user was about to enter the room or not. It was able to anticipate this behavior of the user and directly communicate with other system modules. The purpose of the rule engine was to allow interoperation in the system using simple logical rules, referred here as *smart applications*.

The idea of the prediction in the tracking system was to remove the general system delay so that the interaction with the environment would be pleasant. Rule agents provided backup functions for the light control, in case that the prediction would fail to recognize the behavior. In short, the overall system logic switched the lights on in the room which was entered and off in the room that was left. This happened exactly when the user entered the room without a delay. Additionally, the logic played and paused a TV program based on the user's presence on the sofa. The implemented system modules and especially the

[3]http://www.psychometrica.de/lda.html - Java implementation of the LDA.

different parts of the rule engine are described in more detail in the following subsections. Communication between these different system parts is emphasized. Detailed description of the user tests and measurements are presented in (Järvinen and Vuorimaa, 2012).

6.1 Rule Engine

An oBIX standard based rule engine was developed for the implementation. It is a further development result of our Java based open Facility Management Server (oFMS) (Järvinen et al., 2011). Thus, in addition to the rule management, the rule engine can simultaneously function as an integration platform. However, in this experiment we wanted to use a separate integration platform to follow the proposed architectural design.

The rule engine is not following the ECA structure as there is no separate event part in the rules. Its active component, the smart application executor takes care of regularly updating all the individual device states that are referred in any of the rule conditions. Thus, the conditions are evaluated only when needed. The actual triggering event is a change in any of the device states referred in the condition. The Algorithm 1 illustrates the behavior of the six rules that were used in the experimentation.

Algorithm 1: Rules used in the experimentation.

```
Rule 1: When someone enters or leaves this room, switch on and off the lights accordingly
  if $rulestate.has.changed$ then
    if $rulestate = true$ then
      $lamp1 \leftarrow on$ and $lamp2 \leftarrow on$ and $lamp3 \leftarrow on$
    else
      $lamp1 \leftarrow off$ and $lamp2 \leftarrow off$ and $lamp3 \leftarrow off$
    end if
  end if
Rule 2: When someone enters or leaves this room, switch on and off the lights accordingly
  if $rulestate.has.changed$ then
    if $rulestate = true$ then
      $lamp4 \leftarrow on$ and $lamp5 \leftarrow on$ and $lamp6 \leftarrow on$
    else
      $lamp4 \leftarrow off$ and $lamp5 \leftarrow off$ and $lamp6 \leftarrow off$
    end if
  end if
Rule 3: When the user moves to this room, notify the light batch rules and start observing when the user goes back to the other room
  if $user.at.room.1$ then
    $rule1.rulestate \leftarrow on$ and $rule2.rulestate \leftarrow off$ and
    $rule4.active \leftarrow on$ and $rule4.statechangestolive \leftarrow 1$
  end if
Rule 4: When the user moves to this room, notify the light batch rules and start observing when the user goes back to the other room
  if $user.at.room.2$ then
    $rule1.rulestate \leftarrow off$ and $rule2.rulestate \leftarrow on$ and
    $rule3.active \leftarrow on$ and $rule3.statechangestolive \leftarrow 1$
  end if
Rule 5: When the user sits on the sofa, turn on the TV and start observing when the user leaves the sofa
  if $user.sitting.on.the.sofa$ then
    $tv \leftarrow on$ and $rule6.active \leftarrow on$ and
    $rule6.statechangestolive \leftarrow 1$
  end if
Rule 6: When the user leaves the sofa, pause the program and start observing when the user returns to the sofa
  if $user.not.sitting.on.the.sofa$ then
    $tv \leftarrow off$ and $rule5.active \leftarrow on$ and
    $rule5.statechangestolive \leftarrow 1$
  end if
```

6.2 Smart Application Service

Smart application service offers an access point for external modules to create rules. Following the oBIX principles, it publishes information in the oBIX lobby object, access point to any oBIX based system (Figure 4). After the creation of a new smart application, all the needed references to it are found in the returned smart application oBIX object. This object provides functionality for management of the rule conditions and actions and the logical parameters affecting the rule behavior. Smart application service is able to restore smart applications from the XML presentations stored in the database.

```
<obj href="obix:Lobby">
  <ref name="about" href="about/"
       is="obix:About"/>
  <ref name="smartApplicationService"
       href="smartApplicationService/"
       is="smartApplicationService"/>
</obj>

<obj href="smartApplicationService">
  <op name="make" href="make" in="obix:Nil"
      out="smartApplication"/>
</obj>
```

Figure 4: The smart application service is found in oBIX lobby. The service provides a way to create smart applications.

6.3 Smart Application Executor

Rule conditions can include references to a number of device and object states. The main task of the smart

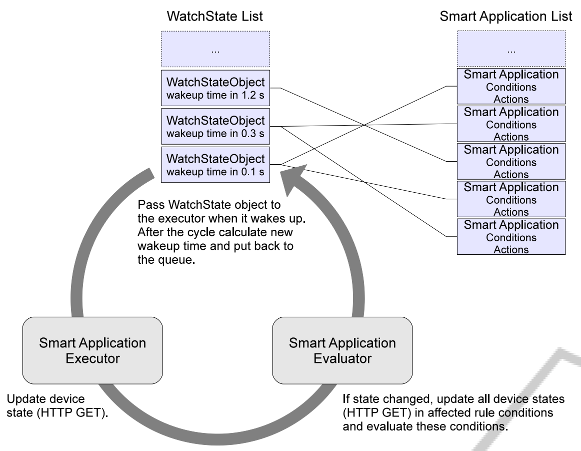
WatchState List
Smart Application List
...
WatchStateObject wakeup time in 1.2 s
WatchStateObject wakeup time in 0.3 s
WatchStateObject wakeup time in 0.1 s
...
Smart Application Conditions Actions
Smart Application Conditions Actions
Smart Application Conditions Actions
Smart Application Conditions Actions
Smart Application Conditions Actions
Pass WatchState object to the executor when it wakes up. After the cycle calculate new wakeup time and put back to the queue.
Smart Application Executor
Smart Application Evaluator
Update device state (HTTP GET).
If state changed, update all device states (HTTP GET) in affected rule conditions and evaluate these conditions.

Figure 5: Smart application executor periodically updates the watchstate objects. If an object state has been changed, it creates a smart application evaluator to evaluate the conditions that refer to this watchstate.

application executor is to take care that these states are kept up to date. Based on the polling intervals defined in the rules, it periodically requests updates for the state values. When any of the states change, it creates a new smart application evaluator to evaluate the rules that include conditions that were affected (Figure 5).

6.4 Smart Application Evaluator

Smart application evaluator is spawned for every change in any of the device states that the smart application executor is updating. Executor passes a list of affected smart applications to the evaluator. Affected smart applications are all the rules that refer to the changed state in their conditions. Evaluator then goes through all the smart applications in this list evaluating their conditions. For the evaluation, it requests state values of the devices. If the return value of the evaluation differs from the current smart application state stored in its XML representation, corresponding rule actions are executed.

6.5 Rule Agent

In practice, rule agent describes a rule as a smart application (Figure 6). An individual rule includes rule properties defining its behavior, a conditions clause, and a list of actions. A conditions clause can include many condition clauses. All watch state rules in an individual condition clause have to be true for the rule condition to be fulfilled. Instead, for a conditions clause to be fulfilled, even one fulfilled condition clause is sufficient. An individual watch state object inside a condition defines a primitive condition for one device state (Figure 7). The resulting simple or-and mechanism can handle basic logic needed for common purposes. For more complex logic, rules can be chained together. This type of chaining is possible because rules are able to refer to the states of each other. Actions list defines the resulting actions that are executed if the state of the smart application is changed.

```
<obj is="smartApplication">
  <int name="stateChangesToLive"/>
  <bool name="active"/>
  <bool name="manualControl"/>
  <bool name="falseStateChange"/>
  <bool name="currentState"/>
  <reltime name="pollInterval"/>
  <op name="makeCondition" in="obix:Nil"
      out="condition"/>
  <op name="deleteCondition" in="obix:ref"
      out="obix:Nil"/>
  <op name="makeWatchState" in="obix:ref"
      out="watchState"/>
  <op name="deleteWatchState" in="obix:ref"
      out="obix:Nil"/>
  <list is="conditions" of="condition"/>
  <op name="makeAction" in="obix:Nil"
      out="action"/>
  <op name="deleteAction" in="obix:ref"
      out="obix:Nil"/>
  <list is="actions" of="action"/>
  <op name="delete" in="obix:Nil"
      out="obix:Nil"/>
</obj>
```

Figure 6: The smart application contract in oBIX. In practice, smart application instances also include href attributes, so that the individual elements can be referred. Thus, each rule and its sub elements have unique URIs.

```
<obj is="watchState">
  <str name="target"/>
  <enum name="relation"
      range="relationshipRange"/>
  <val name="ComparisonValue"/>
  <reltime name="pollInterval"/>
  <obj name="state">
    <bool name="active"/>
    <str name="message"/>
  </obj>
</obj>
```

Figure 7: An oBIX contract for a watch state object defining a simple condition which refers to only one device state. Instances of the contract also include href attributes for each of the elements.

6.6 Integration Platform

We used an integration platform provided by C oBIX Tools (CoT)[4] from Andrey Litvinov. It handles all the

[4]C oBIX Tools is published as an open source project at http://code.google.com/p/c-obix-tools/

basic oBIX features and additionally offers a more efficient technique for polling than the original oBIX specification. The integration platform was used by the lighting system oBIX client, virtual television oBIX client, and, of course, the rule engine and the smart device for the interoperation.

6.7 Smart Device

The tracking system oBIX client was implemented as a smart device. It was able to predict the user entering into a room and managed to compensate the common system delay of approximately one second in that behavior. It used LDA for the prediction when the user was close to the door. LDA considered walking direction and the turning angle as factors to define the behavior. Smart device did not command the lights directly but used the batch rules (Rules 1 and 2) on the rule engine to easily command multiple lights in one room with one request.

7 EVALUATION

The results are evaluated against the requirements for Web services based BA control systems defined in Section 4.

7.1 Description Format

The control system logic is implemented as rules. The system is based on the oBIX standard, which defines an XML-based data description format. The use of oBIX XML data model for the rule description fulfills the requirement R1 per se. This language is used to define rules in a declarative manner by describing them with condition clauses, watch states, actions, and general rule parameters, fulfilling the requirement R2. For further interoperability to exchange rules with other systems, RuleML could be used as a rule description format, and the rule management interface could handle the translation into oBIX for the Web API.

7.2 System Management

In the XML description, a rule element and its sub elements all have unique URIs. Thus, they are manageable through the HTTP oBIX interface. This direct support for URIs as an addressing method simplifies the rule management, provides an access method for the control systems to perceive and control the environment, and facilitates the integration with the rest of the Web. These fulfill the requirements R6 and R7.

7.3 Active Interaction in Distributed Environments

The implemented smart application executor regularly requests updates for the device states that the rules depend on, fulfilling the requirement R5. Additionally, the rule engine contains a smart application evaluator, which evaluates the rule conditions and commands the actions if the conditions are fulfilled. Both the executor during the regular update cycle and the evaluator during the rule evaluation and action commanding, are able to make HTTP requests to Web resources. These fulfill the requirements R3 and R4.

8 DISCUSSION

According to Zelkha, et al. (Zelkha et al., 1998), the ambient intelligence paradigm is characterized by the systems that are:

- Embedded;
- Personalized;
- Adaptive;
- Anticipatory.

The system presented in this paper can be characterized with these features. The integration platform provides a solution for interconnecting the devices embedded in the environment. Functioning of the devices can then be managed and the system personalized through standard BA interface, and automatic behavior be programmed with the rule engine. Utilizing a standard interface enables usage of multiple control systems at the same time, thus allowing modules for adaptive and context aware control to be added to the system. Similarly, the system demonstrates the possibilities for anticipatory behavior with a simple prediction module for lighting control.

We believe that the defined system requirements offer a basis for developing interoperable smart home systems. From the architectural point of view, the main advangement is the support for multiple control systems at the same time. While it offers a possibility for modular system design, it also introduces new challenges for the management of the system functioning as a whole. Cooperative functioning of this kind of distributed applications have been studied in the MAS domain (Weiss, 1999) (Wilmott et al., 2001). Thus, applying advanced MAS techniques and principles for the flexibility, robustness, and conceptual reasoning of the control system agents is one of our future research lines. In contrast to similar existing systems, our solution is based on Web technologies and provides a Web API for the rule management, which allows user agents to create and remove rules, but also to modify the content of existing rules in the rule engine. These with an XML based description format are compatible with Semantic Web techniques and allow us to add a new semantic layer on top of the system in the future, and integrate MASs with the smart home systems to further build intelligence with semantic reasoning.

9 CONCLUSION

We presented a logic layer interoperability solution for distributed smart home systems. The system was based on requirements that ensure the support for features that contribute to the interoperability on the logic layer and enable it on the semantic layer. The implementation considered a rule engine able to observe and control devices through standard Web services interface. Similarly, the rule engine offered a Web services interface for the rule management following the same building automation guideline. The implemented system was functional, fulfilled the requirements, and proved the applicability of the approach. However, the system does not offer all the needed functionalities at the moment. Proper methods are needed for security, authentication, authorization as well as managing conflicts in the logic. As the system is using RESTful Web services, it can partly exploit same well-known techniques that are used in the traditional Web applications to overcome some of the challenges.

We believe that Web services based control systems can offer solutions to gain better interoperability in smart home environments. The interoperability in the system is not achieved by offering a single manner to control the system. It is a joint effort of multiple control systems offering their services openly for managing the logic. Individual control systems can concentrate on their main functions and exploit services from each other, shaping the smart home to a fully functional ecosystem where every service does its own part and interacts with the others.

ACKNOWLEDGEMENTS

Part of the work was done as part of TIVIT Devices and Interoperability Ecosystem (DIEM) project Building Automation work package and supported by The Doctoral Programme in the Built Environment (RYM-TO) funded through the Academy of Finland and the Ministry of Education and Culture.

REFERENCES

Bailey, J., Poulovassilis, A., and Wood, P. (2002). An event-condition-action language for xml. In *Proceedings of the 11th international conference on World Wide Web*, pages 486–495. ACM.

Bernauer, M., Kappel, G., and Kramler, G. (2004). Composite events for xml. In *Proceedings of the 13th international conference on World Wide Web*, pages 175–183. ACM.

Boley, H., Tabet, S., and Wagner, G. (2001). Design rationale for ruleml: A markup language for semantic web rules. In *SWWS*, volume 1, pages 381–401.

Bonifati, A., Braga, D., Campi, A., and Ceri, S. (2002). Active xquery. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 403–412. IEEE.

Bonifati, A., Ceri, S., and Paraboschi, S. (2001). Active rules for xml: A new paradigm for e-services. *The VLDB Journal*, 10(1):39–47.

Chakravarthy, S. and Adaikkalavan, R. (2007). Ubiquitous nature of event-driven approaches: A retrospective view.

Considine, T. and Ehrlich, P. (2006). Open building information exchange (oBIX).

Felser, M. and Sauter, T. (2002). The fieldbus war: History or short break between battles? In *IEEE WFCS*, pages 73–80.

Guinard, D. and Trifa, V. (2009). Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*.

Järvinen, H., Litvinov, A., and Vuorimaa, P. (2011). Integration platform for home and building automation systems. In *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, pages 292–296. IEEE.

Järvinen, H. and Vuorimaa, P. (2012). Anticipatory lighting in smart building. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 390–394. IEEE.

Kay, M. (2007). Xsl transformations (xslt) version 2.0. *W3C Recommendation*, 23.

Leong, C., Ramli, A., and Perumal, T. (2009). A rule-based framework for heterogeneous subsystems management in smart home environment. *Consumer Electronics, IEEE Transactions on*, 55(3):1208–1213.

Lloyd, J. (1994). Practical advantages of declarative programming. In *Joint Conference on Declarative Programming, GULP-PRODE*, volume 94, page 94.

Maile, T., Fischer, M., and Huijbregts, R. (2007). The vision of integrated IP-based building systems. *Journal of Corporate Real Estate*, 9(2):125–137.

Mozer, M. (1998). The neural network house: An environment hat adapts to its inhabitants. In *Proc. AAAI Spring Symp. Intelligent Environments*.

Padgham, L. and Winikoff, M. (2004). *Developing intelligent agent systems: a practical guide*, volume 1. Wiley.

Papamarkos, G., Poulovassilis, A., and Wood, P. (2003). Event-condition-action rule languages for the semantic web. In *Workshop on Semantic Web and Databases*, page 2003. Citeseer.

Papamarkos, G., Poulovassilis, A., and Wood, P. (2004). Rdftl: An event-condition-action language for rdf. In *Proc. of the 3rd International Workshop on Web Dynamics*.

Rutishauser, U., Joller, J., and Douglas, R. (2005). Control and learning of ambience by an intelligent building. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(1):121–132.

Schepers, D., Rossi, J., Höhrmann, B., Le Hégaret, P., and Pixley, T. (2011). Document object model (dom) level 3 events specification. *W3C Working Draft*.

Valtchev, D. and Frankov, I. (2002). Service gateway architecture for a smart home. *Communications Magazine, IEEE*, 40(4):126–132.

Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT press.

Wilmott, S., Dale, J., Burg, B., Charlton, P., and O'Brien, P. (2001). Agentcities: A Worldwide Open Agent Network. *AgentLink News*, Issue 8.

Zelkha, E., Epstein, B., Birrell, S., and Dodsworth, C. (1998). From devices to ambient intelligence. In *Digital living room conference*.