2012 Eighth International Conference on Intelligent Environments

Seamless Integration of Heterogeneous Devices and Access Control in Smart Homes

Ji Eun Kim[1,3], George Boulos[3], John Yackovich[3], Tassilo Barth[4], Christian Beckel[2], Daniel Mosse[3]
[1] Bosch Research and Technology Center, Pittsburgh, PA 15203, USA
[2] Corporate Sector Research and Advanced Engineering, Robert Bosch GmbH, Stuttgart, 70442 Germany
[3] Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15213, USA
[4] Computational Linguistics Department, Saarland University, Saarbruecken, 66123, Germany
jieun.kim@us.bosch.com, gwf5@pitt.edu, jcy8@cs.pitt.edu, tbarth@coli.uni-sb.de, christian.beckel@de.bosch.com, mosse@cs.pitt.edu

***Abstract*—The recent trend of ubiquitous access to embedded physical devices over the Internet as well as increasing penetration of wireless protocols such as ZigBee has raised attention to smart homes. These systems consist of sensors, devices and smart appliances that can be monitored and controlled remotely by human users and cloud services. However, the lack of a de facto communication standard for smart homes creates a barrier against the interoperability of devices from different vendors. We address this challenge by proposing a holistic, extensible software architecture that seamlessly integrates heterogeneous protocol- and vendor-specific devices and services, while making these services securely available over the Internet. Our architecture is developed on top of the OSGi framework and incorporates a semantic model of a smart home system. As a result, we achieve semantic interoperability – the ability to integrate new applications and drivers into the deployed system during runtime. Furthermore, we integrate a new access control model for specific smart home scenarios. As a proof of our concept, we demonstrate the seamless semantic discovery of home devices at runtime by integrating several protocols including X10, Insteon, ZigBee and UPnP into a real test. Using smart phones and cloud services together with our home gateway implementation, we further demonstrate the ease of integration of new applications and drivers.**

Keywords- smart home; interoperation; semantics; access control

I. Introduction

Smart home systems provide automation capabilities that allow home owners to have more complete control over their home, and promote energy efficiency that allows them to save money on energy bills. These smart home solutions integrate many devices having different capabilities such as intrusion detection, video surveillance, fire detection, patient health monitoring and entertainment. Many of these devices use different communication protocols that are mostly incompatible with each other. Examples of such protocols are power line communications like *X10* [1] or *Insteon* [2], wireless communications (*ZigBee* [3], *Z-Wave* [4]), IP-based *UPnP* (Universal Plug-and-Play) [5], SOAP-based web Services on devices such as *DPWS* (Device Profiles for Web Services) [6], Web of Things [7] using RESTful Web services and many more proprietary protocols from diverse manufacturers.

Although the market prediction and current technological trends look promising, we observed that no de facto communication standard exists in the smart home. This hinders the integration of different services (e.g., energy management, security). Therefore, we propose a smart home software architecture based on the OSGi (Open Services Gateway initiative) framework that seamlessly integrates heterogeneous protocols and diverse device types used in home networks. Our aim is to enable end users to add new devices on demand, regardless of the discovery peculiarities imposed by the particular communication protocol. In addition, since device access by applications or end users should not depend on installation details, we introduce an abstraction layer based on a simple semantic domain model.

Furthermore, more devices connected to the home result in more diverse groups of users and services that interact with smart home solutions. This requires a new smart home access control concept. For example, the smart home system should not allow a utility company to access the home owner's health-related data from medical devices. To satisfy this requirement, we propose a new access control model and its implementation supporting our policy model for different users, permissions and multi-attributes including user roles, device type, location, device status and time.

The main contributions of this paper are as follows:

- Analysis of a set of home network protocols with regard to their discovery and integration requirements.
- An extensible home gateway architecture that allows heterogeneous devices to be flexibly installed, managed and accessed during runtime.
- A novel access control mechanism specific to smart home systems.
- The realization of the proposed concept by demonstrating how diverse devices can be integrated and accessed by end users.

The remainder of the paper describes related work in *Section II*, details of driving use case scenarios in *Section III*, our proposed overarching smart home architecture in *Section IV* and the home gateway architecture including device stack, discovery as well as usage of devices in *Section V*. We present the access control mechanism in *Section VI* and our demonstrator in *Section VII*. In *Sections VIII* and IX, we give an outlook to future work and conclude our paper.

978-0-7695-4741-1/12 $26.00 © 2012 IEEE
DOI 10.1109/IE.2012.57

IEEE computer society

Authorized licensed use limited to: MIT. Downloaded on September 26,2024 at 18:49:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1010
Microsoft Corporation v. Edge Networking Systems, LLC

II. Related Work

Related work is categorized into the research areas of home network standards, smart home architectures targeting interoperability, and smart home access control mechanisms. In the following, we discuss each of these aspects separately:

Home network standards: Prevailing home network standards use different communication media (e.g., power line, various RF bands), device address schemes of static or dynamic addresses, and different device discovery mechanisms. Low data rate protocols such as X10 and Insteon do not provide device descriptions. ZigBee, KNX [8] and EnOcean [9] provide device descriptions in the form of standardized profiles specific to domains such as home automation. The UPnP and DPWS standards use XML to describe device information. Semantic descriptions are not provided by any of these home networks. Security mechanisms of these protocols vary in behavior. Examples are pairing mechanisms of devices, checksums of message payload and data encryption techniques using symmetric keys. Due to these diverse and often incompatible mechanisms from different network standards, smart home systems in the market remain fragmented and provide only partial solutions addressing single protocols and subsets of devices.

Smart home architectures: Despite this heterogeneous nature of home network standards, existing smart home research generally assumes a homogeneous underlying architecture. MavHome [10], for example, predicts activities in a home and makes the home act as an intelligent agent providing optimal support for its inhabitants. In [11] the authors address the important role of context for smart home applications by providing adaptive middleware and APIs that provide context to applications. Projects by Kawsar et al [12] and Zhang and Brugge [13] aim to assist end users to build their own individual smart home applications. While these visions are important for the success of smart homes, dealing with device and service heterogeneity is a crucial requirement for them to be realized.

There are a number of similar architectural approaches that target interoperability on the network protocol level, for instance, DOG (Domotic OSGi Gateway) [15], Hydra [19] and a project by Peláez [30]. All of them are based on OSGi and use a semantic model for abstraction. The device interoperability is achieved by a multi-layer device stack which roughly consists of drivers, common adapters for similar device types and a high-level representation described semantically by a domain model. However, none of them gives a clear account on how devices communicating over different protocols are discovered. On the contrary, we present a complete device discovery workflow: plug and play of heterogeneous devices during runtime, extensibility to new devices that are not foreseen during system development, and the use of a barcode reading functionality with smart phone cameras as a unique approach to improve usability.

Moreover, our approach integrates cloud services to increase coverage of the device discovery process and to extend the smart home functionality with new applications, drivers, and computationally intensive services. The authors of [11] present an architecture to integrate cloud services and smart home networks. However, unlike us, they neither implement the architecture in a prototype nor employ it directly in a user interface.

Access control: The authors in [20] provide RBAC (Role Based Access Control) for the OSGi service environment and use XACML (eXtensible Access Control Markup Language) [21] for their policy descriptions. However, the authors do not address other important attributes in our policy model, such as device type, location, device status and current time.

In summary, previous work covers specific attributes of smart homes such as communication protocols, semantic description of devices, access control schemes, and user interaction. The contribution of this paper is to investigate a *holistic extensible smart home architecture* that enables these concepts to play together and to prove it with our prototypical implementation.

III. Driving Use Case Scenarios

A smart home user study [29] uncovered that high cost of ownership, inflexibility, poor manageability and difficulties in achieving security constitute four barriers for broad adoption of home automation. We find that the heterogeneous nature of the smart home system as well as the stakeholders is an intrinsic cause of these barriers. This section describes three scenarios that are crucial to our smart home solution.

First, home owners often add home devices incrementally over time due to limited budgets and innovations in the market. Therefore, the combined process of *seamless plug and play of devices and discovery of semantic services is necessary to allow flexibility in managing smart home systems.* In our scenario, a home owner brings a new device (e.g., Insteon dimmer light) to the home. She uses her smart phone camera to scan a barcode of the device. Then the smart home system provides a corresponding discovery wizard for the device installation. The smart home system connects to the application store in the cloud in order to download relevant software drivers and basic applications recommended by the system. Upon the home owner's approval the software can be deployed on the smart home system. Once this discovery process is completed, the home owner is able to access the discovered device remotely using his/her smart phones or other user interface devices.

Second, *new smart home applications should be developed without detailed knowledge* about devices and protocol-specific information. Instead, the smart home system should allow the applications to rely on an inference capability. Assume that a developer wants to develop an application to provide a service to turn off all lights on the first floor of a smart home if there is no one present for more than a certain period of time. Since every home has different configuration of devices types, the developer should be able to get a list of available devices with a certain level of abstraction. In our

Authorized licensed use limited to: MIT. Downloaded on September 26,2024 at 18:49:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1010
Microsoft Corporation v. Edge Networking Systems, LLC

scenario, the developer would request a list of the available lighting devices using the abstract interface and get the corresponding device states to determine the occupancy status without knowing the device address, protocol and other configurations.

Third, our smart home system connects many different devices, which can be used by diverse users. The type of users include adults and children in family, visitors such as babysitters, and service providers (e.g., utility companies and professional healthcare givers) accessing smart home remotely in form of services. This requires *suitable authentication and authorization for different users and devices*. We provide a mechanism that enables the home gateway to authenticate different roles of users and authorize their different requests. For example, we address the need to prevent a healthcare giver from accessing the home owner's non-healthcare related devices. In addition, our scenarios include access control that utilizes device status, device location and current time as access control attributes.

IV. Overarching Software Architecture

Our smart home system (Figure 1) consists of a *home gateway*, which connects different types of home devices and provides standard interfaces that are accessible through web services by using smart phones or any web-based user interface devices. Home devices are categorized into *end devices* and *controller devices*. *End devices* are sensors and/or actuators, such as temperature sensors, video cameras, motion sensors, smart appliances, plug-in modules or any devices that provide some direct smart home functionality. *Controller devices* do not offer specific services within the home. Instead they are gateway-type devices that allow the *home gateway* to communicate with *end devices*. They typically connect via USB or Serial ports to the *home gateway*. Examples of these are modems such as the Insteon PowerLinc model 2413s (RS-232), X10 Active Home USB controller, and the Digi XStick Xbee USB adaptor. For protocols like UPnP, which work on top of IP, the *controller device* aspect is more implicit. The only hardware required to communicate with an UPnP device is a network card, which can be the *controller device*.

The *home gateway* connects to a *cloud solution center* that holds an *application store* and provides *cloud services*. The *application store* manages the dependencies of software and home devices, and downloads software drivers and applications to the user's *home gateway*. A typical cloud service, for example, is a video surveillance service requiring

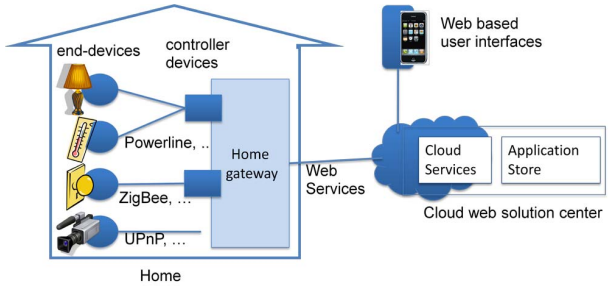

Figure 1: High-level architecture for the smart home

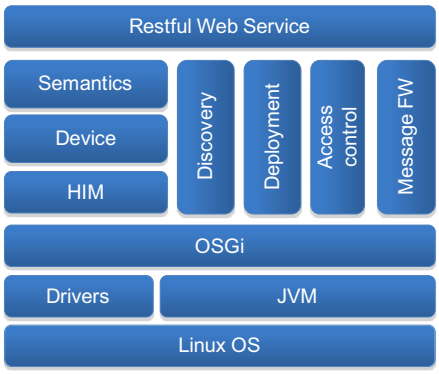

Figure 2: Core software building blocks of Home Gateway

large amounts of storage.

V. Smart Home Gateway Architecture

Important architectural requirements for our smart home system are *interoperability* and *dynamic integration* of many types of drivers and devices. Different types of devices and services can be added and removed seamlessly during runtime, and services within the system need to discover the existence of other services with which they need to interact. To tackle this problem, we build our system on top of the Java-based OSGi framework, which enables services to be plugged in and out at runtime. The *OSGi device access service* facilitates the process of plug and play of devices and services through the device attachment phase. The *OSGi event admin service* is responsible for the communication (e.g., "new device registered") between components while keeping implementations independent of one another. Figure 2 illustrates the software building blocks of our smart home gateway. Our architecture employs a device stack and a dynamic message handling framework that processes commands in different ways depending on the context. The *HIM* (Hardware Interface Manger) detects any controller device that is connected to the system hardware. The *Deployment Manager* connects to the application store in the cloud and forwards to it properties of controller devices discovered by the *HIM* in order to download proper drivers for new devices. It manages installation of software and update of the home gateway profile in the cloud. *Access Control*

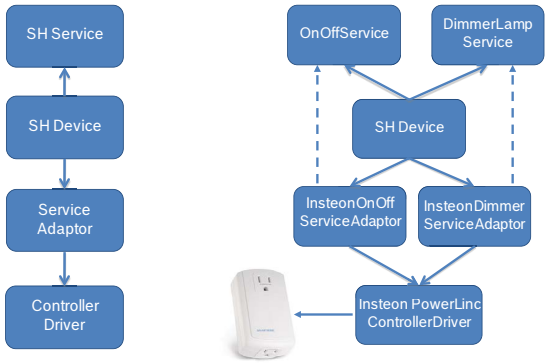

Figure 3: Device Stack (a, left) and an instance of Insteon device (b, right)

Authorized licensed use limited to: MIT. Downloaded on September 26,2024 at 18:49:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1010
Microsoft Corporation v. Edge Networking Systems, LLC

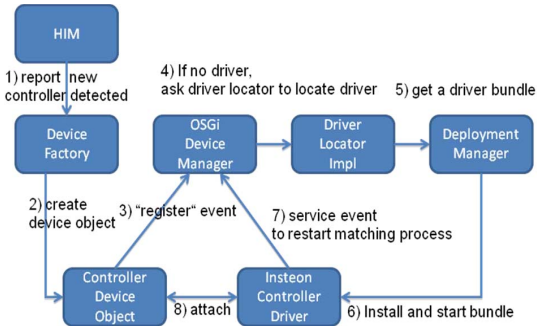

Figure 4: New controller discovery

manages authentication and authorization for different requests. Remote access is available only through *Restful* web services.

A. Smart Home Device Stack and Discovery

Our smart home device stack (Figure 3-a) allows for uniform services to access diverse types of devices. *SHService* (*SmartHomeService*) represents a service provided by a device in a protocol-agnostic way. *SHDevice* (*SmartHomeDevice*) represents a proxy to a physical end device. This object contains common home device information such as the device's URI (Universal Resource Identifier), device type (e.g., video camera), services supported by this device, protocol information and device location. *Service Adaptor* is specific to a protocol implementation for *SHService*. *Controller Driver* represents the features available by the controller device, including the low-level handling of commands. It is responsible for either acting as the base driver or communicating with lower-level hardware drivers such as USB and Serial communication to send and receive information from the device.

Figure 3-b shows an example of the device stack. The Insteon device's two capabilities of turning on/off and dimming the lamp are represented as two protocol-agnostic services of *OnOffService* and *DimmerLampService*. *InsteonOnOffAdaptor* and *InsteonDimmerLampAdaptor* are Insteon protocol-specific implementations to turn on/off and dim the lamp device. *InsteonPowerLinc ControllerDriver* is an implementation of the Insteon controller driver which sends commands through USB communication.

New Controller Device discovery: We describe the controller discovery through an example illustrated in Figure 4. Upon the plug-in of a controller device (such as the PowerLinc RS232), the *HIM* detects a new controller and sends controller device information to the *DeviceFactory* module. The *DeviceFactory* module, which is responsible for the life cycle of devices, creates a *Controller Device Object* with device information, and registers a "*new device*" event. The *OSGi Device Manager* detects this event and starts the matching process of *Controller Device Object* and the corresponding driver (*Controller Driver*). If no driver is found, the *OSGi Device Manager* contacts *Driver Locator,* an OSGi Service interface for locating drivers for a device. Our implementation of *Driver Locator* calls *Deployment Manager*, which is responsible for downloading and installing the required software bundles from the *application store* in the cloud. The *application store* manages software dependencies of the connected devices and other properties such as cost of software packages. Once the installation is complete, *Device Driver* registers a *Controller Driver* and *Device Manager* restarts the matching process, and *Controller Device Object* finally matches with *Controller Driver.*

New End Device discovery: Each protocol has its own device discovery mechanism. We categorize discovery mechanisms into three types: *manual*, *semi-automatic* and *automatic device discovery*.

Manual discovery is device discovery that must be done entirely outside of the protocol specification. One example is X10, which provides no means for automated discovery of devices. In these instances the home gateway must derive all information about the device via user input. *Semi-automatic discovery* applies to protocols that support some level of device discovery using the protocol, but still require human involvement. Insteon is an example of semi-automatic discovery in that the user is required to either 1) input the hardware address of a device to discover it, or 2) press a hardware button on the device to initiate a discovery mode. The *ZigBee* protocol also falls into this category of home automation profiles.

Automatic discovery does not require any user interaction unless the system wants to get additional (non-protocol) information specific to the user's environment. An example of automatic discovery is the UPnP protocol which detects and adds new devices automatically to the network system. In our system, we ask users to confirm the automatically discovered device and ask them to provide semantic information such as location and a user-friendly name.

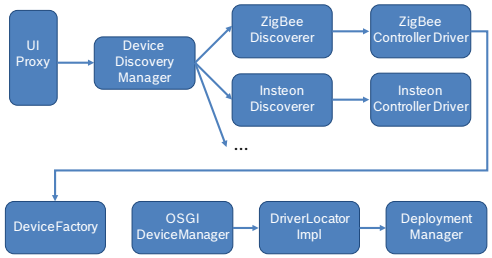

Figure 5: ZigBee End device Discovery

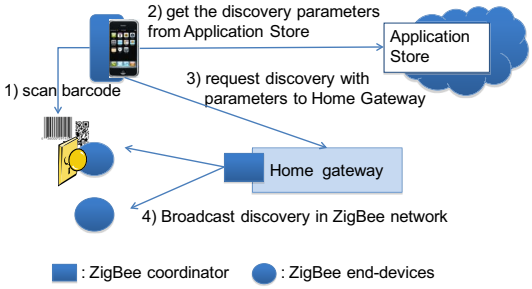

Figure 6: ZigBee End-Device Discovery Process

Authorized licensed use limited to: MIT. Downloaded on September 26,2024 at 18:49:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1010
Microsoft Corporation v. Edge Networking Systems, LLC

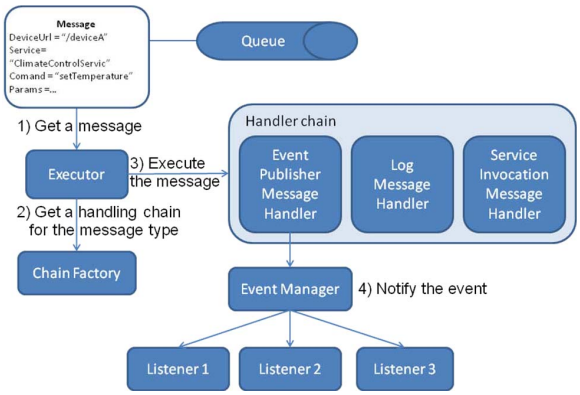

Figure 7: Message Framework

Manual discovery and *semi-automatic discovery* rely on user interaction to add new devices to the smart home system.

To mitigate the complexity of the discovery process, our architecture utilizes a camera on a smart phone device, which is capable of scanning the barcode of devices and sending the barcode to the *application store* which manages the discovery parameters specific to the device. Depending on the discovery parameters returned by the *application store*, we provide users with a corresponding wizard to guide them to easily add the new device.

At the home gateway, we introduce *Device Discoverer* modules which are specific to each protocol. Depending on the input from the smart phone, the home gateway calls the corresponding discoverer to add the new device.

Figure 5 and Figure 6 illustrate an example of new ZigBee end device discovery. In our scenario, the user does not know which discovery mechanism ZigBee supports. She scans the barcode of the ZigBee end device with her smart phone camera which triggers the proper wizard to discover the new device. The smart phone sends the discovery request with parameters from the *application store* in the cloud to the *home gateway*. The *home gateway* finds the corresponding discoverer which sends a broadcast message to the local ZigBee network. This is equivalent to pushing a button on the controller, through *ZigBeeController Driver*. Once a new device is discovered, similar to the controller device discovery process, *Device Factory* creates a new device object (*SHDevice*) and *OSGi Device Manager* matches the corresponding *Service Adaptors* for that device. *Service Adaptors*, in addition to matching to *SHDevice object* instances, must be assigned to a *Controller Driver* in order to execute commands.

B. Message Framework

We introduce a smart home internal message (*SHMessage*) to communicate with different software modules in the home gateway. *SHMessage* can represent various types of framework messages, such as command invocation messages (e.g., turn on kitchen lights), or state change messages (e.g., data updates from a motion sensor if it detects motion). To provide a uniform way of accessing the *SHMessage*, the concept of *Message Helper* is provided. *Message Helper* adopts the *Decorator* design pattern [22] that encapsulates the *SHMessage* object and provides methods to access its fields. *SHMessages* are handled in a centralized yet dynamic way using a mixture of *Chain of Responsibility* and *Strategy* design patterns. A *SHMessage* is handled by invoking a list of *Message Handlers*, each of which is a logical entity responsible for executing a specific task such as raising an event throughout the system or logging the message. The list of *Message Handlers* is constructed on-the-fly by the *Chain Factory* module, which uses the message type to derive the purpose of the *SHMessage* and how it should be handled. In this way we provide an extensible framework where handlers may be added and removed dynamically.

Figure 7 illustrates how our message framework works for an example service. A web service creates a *SHMessage* with priority and timestamp to invoke an action on a device (in this example setting the temperature value) and inserts it in the queue. The *Chain Factory* then constructs the chain of *Message Handlers* that will handle the message. In this example, *Event Handler* raises a new event to any modules that are interested in being notified of this particular type of occurrence (the set temperature command). Following this the *LoggingHandler* handles the message, which will record the execution of the message in the log. Finally the *ServiceInvocationHandler* carries out the message's purpose by sending the message's intended command to the appropriate *ControllerDriver*.

C. Semantic Integration

Our architecture incorporates a semantic model that enables new applications to be developed independently from the concrete environment in which they are deployed. The semantic model is formalized as an ontology. Ontologies translate a domain of interest into a set of *concepts*, *properties* and *relations* governed by strictly formalized semantics. Automatic inference operations can draw implicit conclusions from the explicitly stated knowledge.

Consider a typical yet simple example application such as "turn on all lights on the first floor". The application will use an abstract "lighting" super-type to retrieve all possible lamp devices. Moreover, based on the specified device location and the contextual knowledge (e.g., kitchen is on the first floor), it has to infer that devices located in the kitchen are also located on the first floor. After analyzing such examples, we consider the following properties essential for the semantic knowledge in the smart home domain and its potential use by new applications:

- Taxonomy of home devices and home environment to allow for abstraction and on-the-fly semantic device retrieval, together with contextual knowledge (e.g., location and time) required for context-aware applications.
- Reasoning on the knowledge base to infer implicit knowledge from a minimum number of explicit facts.
- APIs to retrieve and modify semantic knowledge without knowledge about formal ontology descriptions.

Authorized licensed use limited to: MIT. Downloaded on September 26,2024 at 18:49:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1010
Microsoft Corporation v. Edge Networking Systems, LLC

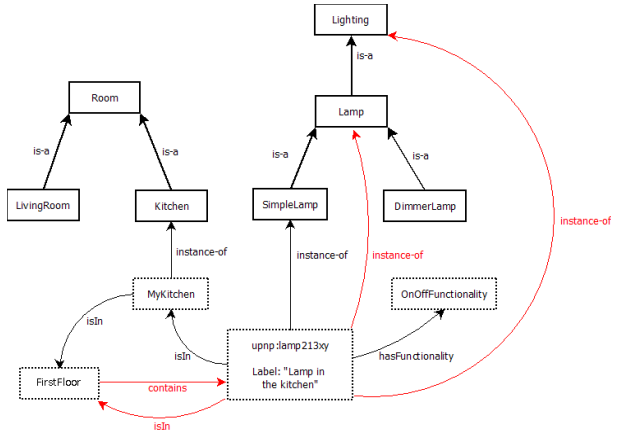

Figure 8: Part of the ontology (red arcs: inferred relations; dashed boxes: knowledge for a smart home instance; solid boxes:generic domain knowledge)

Knowledge on the smart home system has been addressed by domain ontologies such as *DogOnt* [15] and *Hydra* [19]. We build our semantic model on top of *DogOnt*. Figure 8 shows an excerpt from our adapted domain ontology which supports the example application presented above: transitive location properties (*isIn* or *contains)* enable contextual reasoning. The concrete lamp device instance is grounded in the multi-level device taxonomy and connected to the services it offers, which allows for abstract semantic device retrieval.

We formalized the ontology using the Web Ontology Language (OWL) [18], because it is well supported by toolkits and reasoning engines. Smart home applications require fast responses, making efficient reasoning engines especially important. Such high performance reasoning can be achieved when reducing the expressivity of the model by excluding certain constructs from OWL. In order to benefit from large expressivity and also perform fast reasoning we use different components for offline (consistency checking, device instance classification) and online (knowledge retrieval) reasoning.

Online reasoning along with data storage is handled by *OWLIM* [23], a fast and scalable reasoning engine. However, OWLIM excludes some constructs important to automatically detect an inconsistent model (such a model might indicate a configuration mistake). One example is disjointness of device classes, which prevents a device from being simultaneously configured, e.g., as a lamp and a TV. Thus, we use the more powerful *Pellet* [24] reasoning engine to guarantee the consistency of the model and fully classify device instances during discovery and gateway startup.

We provide a common Java-based interface to these engines, with simple methods to retrieve and modify device instances. We support location, device class and device ID as filter options. Within this framework, there should be no need for application developers to be proficient with OWL-related formal methods.

VI. ACCESS CONTROL FOR THE SMART HOME

In this Section we propose a novel access control mechanism for the smart home.

A. Policy Model

Our policy model includes different *roles* (e.g., administrator, adult, kid). A user can be a member of many roles. The policy model has four *permission types*: user management, device management, controlling device, and monitoring device. *User management* permission represents the ability to add, remove and modify user roles. *Device management* permission represents the ability to add, remove and modify devices. *Controlling device* permission represents the ability to issue a command to a device such as opening a door lock or turning on a light. *Monitoring device* permission represents the ability to get state information from sensor devices, which includes getting temperature feed, getting heart monitor feed, video streaming.

A smart home policy model represents fine-grained access policies similar to the ones we enforce in real life. We identified four *attributes* capable of expressing a wide range of rules when combined. The *device type* attribute limits the access to a given device type (e.g., light devices). The *location* attribute determines access to the devices in a specific location. It also limits access to a device by restricting user location at access time. *Device status* limits the access to a device by restricting the status of the device at that time. The *time* attribute limits the access to a device for a (possibly recurring) period of time. The policy maps a user *role* with *permissions* on a device with a collection of *attributes*.

The *device type* attribute is important in the smart home domain, because our system interacts with different user roles (e.g., utility company, healthcare provider) from outside the home with limited access. Also, a family usually consists of different age groups which often require different access policies for safety or other purposes. The *location* attribute is useful for more fine-grained control. A house resident can limit the access of some places within the home for visitors. For example, a visitor may be allowed to control devices in the living room, but not in the bedroom. *Device status* is also required especially concerning future smart grid integration. One possible scenario in the future smart grid is that the service provider should not turn off the laundry machine while the machine is on. The *time* attribute provides users with a flexible way to control access during different times of the day, such as common parental control schemes. For example, parents may only allow their kids to access entertainment devices (e.g., television) during specified time durations.

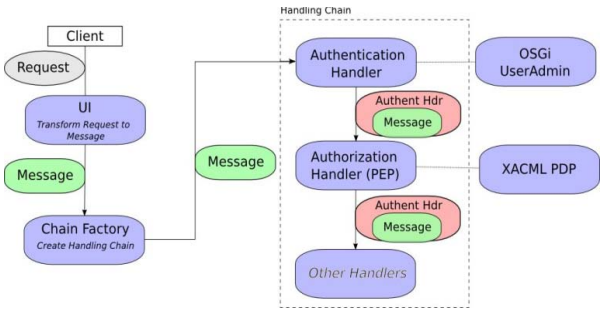

Figure 9: Access Control Design Concept

Authorized licensed use limited to: MIT. Downloaded on September 26,2024 at 18:49:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1010
Microsoft Corporation v. Edge Networking Systems, LLC

```
<Rule RuleId="Rule_6" Effect="Permit">
<Target><Subjects><Subject>
<SubjectMatch MatchId="string-equal">
  <SubjectAttributeDesignator DataType="string" AttributeId="group"/>
  <AttributeValue DataType="string">Kids</AttributeValue>
</SubjectMatch>
</Subject></Subjects>
<Resources> <Resource>
<ResourceMatch MatchId="string-equal">
   <ResourceAttributeDesignator dataType="string" AttributeId="domain"/>
   <AttributeValue DataType="string">Entertainment</AttributeValue>
</ResourceMatch>
<ResourceMatch MatchId="string-equal">
   <ResourceAttributeDesignator DataType="string" AttributeId="status"/>
   <AttributeValue DataType="string">On</AttributeValue>
</ResourceMatch></Resource></Resources>
<Actions> <Action>
<ActionMatch MatchId="string-equal">
  <ActionAttributeDesignator DataType="string"AttributeId="action-id"/>
  <AttributeValue DataType="string">control-device</AttributeValue>
 </ActionMatch>
 </Action></Actions> </Target>
<Condition FunctionId="time-less-than-or-equal">
  <Apply FunctionId="time-one-and-only">
    <EnvironmentAttributeDesignator DataType="time" AttributeId="current-time"/>
  </Apply>
  <AttributeValue DataType="time">19:00:00</AttributeValue>
</Condition>
</Rule>
```

Figure 10: Example of XACML Policy (shortened)

B. Access Control Architecture and Design

To realize access control within our architecture, we develop a hybrid approach of *OSGi User Admin service* and *XACML*. While the *OSGi User Admin service* is moderately expressive, it does not allow us to express the multitude of variables introduced in our policy model. Thus, we use it only to represent subjects (user roles) and assist in our authentication process, while *XACML* specifies and enforces policy given subjects. Figure 9 illustrates how access control concepts are implemented in our architecture. The client request is transformed into new internal messages (*SHMessage* discussed in Section *V.B*). Each message is wrapped with an authentication header which contains a flexible form of user credentials such as username/password or an authentication token. The message is then enqueued within the message framework for execution. *AuthenticationHandler* consults the *OSGi User Admin service* to validate given credentials and get the user's roles. The *AuthorizationHandler* fulfills the role of *PEP (Policy Enforcement Point*) as specified by *XACML*. It checks the authorization object of the message (representing the subject), the intended action and the object of the message, as well as other attributes such as location and time, and submits an *XACML* query based on this information to *PDP (Policy Decision Point*). *PDP* evaluates the *XACML* policy to see if the intended action (e.g., turn on an entertainment device) is indeed an authorized one. If the policy allows the request, the *PDP* will refer the message to the next handler for further processing. If the policy does not permit the action, the message is denied and dropped.

Figure 10 is an example of the *XACML* policy used in our demonstrator, permitting kids to turn on entertainment devices only before 7pm.

VII. Proof of Concept

We demonstrate our smart home architecture and design by implementing a prototype that consists of an actual home gateway, real devices and a smartphone as a user interface. The prototype realizes discovery of selected new devices and integrates services based on semantic information. Our access control concept is also demonstrated with example policies covering selected users, devices and services.

The home gateway prototype is running on Ubuntu 9.04, supporting the OSGi implementations of *ProSyst* [25] and *Equinox*. It integrates devices based on *X10*, *Insteon*, *ZigBee* and *UPnP*. We have integrated a variety of different devices: sensors (motion, water leak), on/off adapters for home appliances and dimmer lamp adapters, RF transmitters to control TV, and video surveillance cameras. For the user interface devices, *iOS* and *Android* devices are used. The cloud solution center currently supports the *application store* component, which is deployed in our private cloud. Figure 11 shows our demonstrator with selected devices.

During runtime, the user connects X10, Insteon and ZigBee controller devices via USB to the home gateway. The home gateway then detects the presence of new controller devices and creates new *controller device objects*. The UPnP drivers discover a new UPnP video camera when it is plugged into the network and send a notification to the user's smartphone (*automatic discovery*). For other end devices, our demonstrator uses iOS Android native apps to start discovery of end devices by scanning the barcode of the devices. The devices are discovered *semi-automatically* and *manually* as discussed in *Section V.A*.

We demonstrate the message passing and the semantic abstraction layer by implementing a specific application. The user can use her smartphone to turn on and off devices of a certain class (*Light*, *Entertainment*) located in a certain room (*GuestRoom*, *Kitchen*). This sample application was implemented by a developer without particular knowledge about OWL-related technologies, using only our interface to the semantic model.

The demonstrator also contains a policy model to show

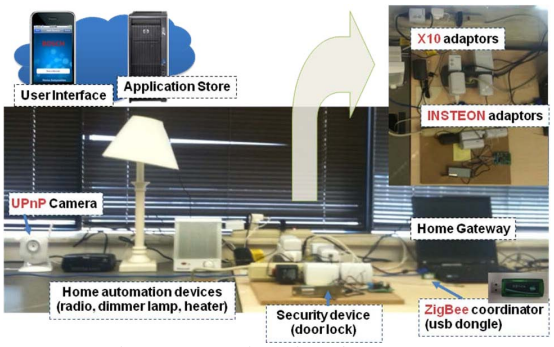

Figure 11: Overview of our demonstrator

Authorized licensed use limited to: MIT. Downloaded on September 26,2024 at 18:49:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1010
Microsoft Corporation v. Edge Networking Systems, LLC

access control for the different users, permissions and attributes discussed in *Section VI*. Users can remotely access discovered devices using smart phones from anywhere based on the access control policy. Our example policy grants all permissions to the users having the *adult* role, while it restricts accesses for the users belonging to the *kid* role. Example policies include "kids are allowed to control all lighting devices in the guest room" and "kids are allowed to turn on entertainment devices before 7pm". We use *a permit-override algorithm*, only allowing requests which match a policy rule, and rejecting all others.

VIII. Future Work

The smart home gateway connects various safety-related devices and the misuse of connected devices may harm the quality of devices and services. For example, a simple service to increase temperature by one degree every minute will lead to failure of a heating device unless the device itself supports suitable protection mechanisms. Therefore, the protection mechanisms for such use cases should be addressed in our existing architecture. Furthermore, we will provide a mechanism for conflicts detection and resolution of different intents that access the same device at the same time in order to provide safe and reliable smart home systems.

In addition, we believe usable and useful interfaces for home owners to specify access control policies to be an important improvement to be investigated.

IX. Conclusion

This paper proposes an extensible OSGi-based architecture for highly heterogeneous smart home systems to enable dynamic integration of devices and services. We demonstrate how the interoperation of different home network protocols can be handled in a systematic way by corresponding device discovery approaches (manual, semi-automatic and automatic). Cloud services and smart phone user interface that provide driver resources and discovery wizards make it easier for users to install new devices. A flexible messaging system together with a semantic abstraction layer supports application development. Finally, our access control policy gives home owners robust control over the way users can access their devices. We believe this work will lower the barriers for smart home systems to become amenable for broader adoption.

Acknowledgment

We thank many researchers in Bosch Corporate Research for discussion of key requirements for smart home systems and Professor Adam Lee at University of Pittsburgh for discussion and review of our access control proposal.

References

[1] X10 PowerHouse, Tech. Rep., 2001, Available: http://www.x10.com.
[2] Smart Home Technology, "INSTEON - the details", 2005, Available: http://www.insteon.net/pdf/insteondetails.pdf
[3] ZigBee Alliance, "ZigBee Home Automation Public Application Profile", 2010
[4] Z-Wave Alliance, Available: http://www.z-wavealliance.org/
[5] UPnP Forum, "UPnP Device Architecture v.1.1", 2010, Available: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf
[6] D. Driscoll, A. Mensch, "OASIS device profile for web services (DPWS) Version 1.1", 2009, Available: http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01
[7] D. Guinard et al., "Towards physical mashups in the Web of Things", Proceedings of the 6th International Conference on Networked Sensing Systems (INSS), 2009 , pp 1-4
[8] KNX Association, "KNX Specifications", Available: http://www.knx.org
[9] EnOcean Alliance, "EnOcean Equipment Profiles (EEP) v2.0", 2009
[10] Diane J. Cook et al., "MavHome: An Agent-Based Smart Home", Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003, pp 521 – 524
[11] M.Huebsher and J.McCann, "Adaptive middleware for context-aware applications in smart-homes", Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, 2004. doi: 10.1145/1028509.1028511
[12] F. Kawsar et al, "Deploy spontaneously: supporting end-users in building and enhancing a smart home", Proceedings of the 10th international conference on Ubiquitous computing, 2008. doi: 10.1145/1409635.1409673
[13] T. Zhang and B.Brugge, "Empowering the user to build smart home applications", Towards a Human Friendly Assistive Environment, IOS Press, 2004, pp 171- 183
[14] OSGi Alliance, "OSGi Service Platform Core Specification v 4.2", 2009, Available: http://www.osgi.org
[15] D. Bonino et al., " The DOG Gateway: Enabling Ontology-based Intelligent Domotic Engironments", IEEE transactions on consumer electronics, 2008, vol. 54/4, pp 1656-1664
[16] The Eclipse Equinox project, Available: http://www.eclipse.org/equinox.
[17] D. Bonino and F. Corno , "DogOnt - Ontology Modeling for Intelligent Domotic Environments", The Semantic Web - ISWC 2008, Lecture Notes in Computer Science, 2008. doi: 10.1007/978-3-540-88564-1_51
[18] D. McGuinness and F. van Harmelen, "OWL Web Ontology Language Document Overview", W3C recommendation, 2004
[19] M. Eisenhauer et al., "A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence System", IEEE SECON Workshops , 2009, pp 1-3
[20] G. Ahn et al., "Towards Role-based Authorization for OSGi Service Environments", Proceedings of IEEE International Workshop on Future Trends of Distributed Computing Systems, 2008, pp 23-29.
[21] OASIS eXtensible Access Control Markup Language (XACML), Available: http:// www.oasis-open.org/committees/xacml/.
[22] E. Gamma, R. Helm , R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Professional, 1994
[23] A. Kiryakov, D. Ognyanov, D. Manov, "OWLIM – A pragmatic semantic repository for OWL", Web Information Systems Engineering – WISE 2005 Workshops, Lecture Notes in Computer Science, 2005, Volume 3807/2005, pp 182-192
[24] E. Sirin, B. Parsia, B.Grau, A. Kalyanpur, Y. Katz , "Pellet: A practical OWL-DL reasoner", Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 5, Issue 2, June 2007, pp 51-53, Elsevier
[25] ProSyst OSGi services, Available: http://www.prosyst.com
[26] Sun's XACML Implementation, Available: http://sunxacml.sourceforge.net/
[27] THJ Kim et al., "Challenges in Access Right Assignment for Secure Home Networks", Proceedings of the 5th USENIX Workshop on Hot Topics in Security, 2010
[28] W. Zhiqiang et al., "Research and design architecture of cloud architecture for smart home", IEEE International Conference on Software Engineering and Service Sciences, 2010, pp 86-89
[29] A. Brush et al., "Home automation in the wild: challenges and opportunities", 2011 Proc. of the annual conference on Human factors in computing systems (CHI 2011), 2011, pp. 2115-2124
[30] V. Peláez et al., "Multilevel and Hybrid Architecture for Device Abstraction and Context Information Management in Smart Home Environments", Ambient Intelligence: First International Joint Conference, 2010

Authorized licensed use limited to: MIT. Downloaded on September 26,2024 at 18:49:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1010
Microsoft Corporation v. Edge Networking Systems, LLC