Cloud Computing for Home Automation

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Clemens Pühringer
Registration Number 1026571

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ.Prof. Dr. Wolfgang Kastner
Assistance: Dipl.-Ing. Markus Jung

Erklärung zur Verfassung der Arbeit

Clemens Pühringer
Kirchenstrasse 11, 4961 Mühlheim

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Author)

Abstract

Home automation and the Internet of Things are important fields of research which are thought to gain a lot more public attention in the years to come. Cloud computing has the potential to provide easy access to home automation for the general public by providing easy to use online services. Open and standardised protocols for home automation devices further increase the convenience by offering more choice and freedom to the customer.

In the course of this thesis, state-of-the-art communication technologies and cloud services in this field are presented and compared based on their features. Furthermore, benefits and drawbacks of cloud-based home automation are discussed and evaluated with regard to cost and security. To show the basic concept of cloud-based home automation, an API is presented that can be integrated into the Appscale Platform as a Service. The API is able to communicate with remote devices via CoAP and oBIX and can be used like any existing API in Appscale.

Contents

List of Figures

CHAPTER 1

Introduction

In recent years, the price for small electronic devices has dropped significantly. This development towards cheap embedded devices drives forward the idea of ubiquitous computing, where humans are surrounded by a multitude of such devices to make their lives easier. Naturally, this also includes the living space of humans, mainly their homes. These can be extended by electronics to automate certain routines (e.g. making coffee in the morning) or just to enable some services to be remotely controlled by the owner (e.g. moving sunblinds via a smartphone). The term *smart home* has taken hold in conjunction with such automated households. There are already a lot of vendors offering such services and devices, but a large problem so far has been the heterogeneity of the different systems offered by the vendors. Of course, every company wants to promote their own product, but the difference between the technologies often leaves the owner no choice but to buy additional devices directly from the original vendor or risk wasting money/time on an incompatible device. The lack of a global standard for home automation devices and the eventual need to create such a standard lead to technologies which are already established standards in today's world. IP based networks seem the most promising when trying to interconnect a lot of devices. The main problem, that the devices do not have enough power, disappears as ever more small and powerful devices are created. The trend is going towards IP and a lot of research today is centered toward an IP based Internet of Things, an example of this new research trend is ZigBee IP [1]. By utilizing the (albeit slowly) emerging IPv6 protocol stack to supply every device with a unique address, it will also be possible to connect every device to the Internet without the immediate need for NAT or a mediating entity like a gateway. This approach removes the need for proprietary field bus systems. What remains is the need for an open common transport and application protocol so that devices can be replaced and setups extended without fearing that the new device will not be compatible. In conjunction with cloud computing, which becomes more and more integrated into our lives, home automation can be taken to the next level. The *smart home* can send data into the cloud where it can then be processed for the user's needs. The cloud can also assume the role of the middleware or gateway, mediating beween proprietary protocols and formats and presenting the information to the user in a standardised way. Cloud computing offers much more convenience compared to local Web servers or control panels, and so it seems probable that this will be the prevailing technology in connection with home automation. In Figure 1.1 the basic principle of home automation in a cloud environment is shown.

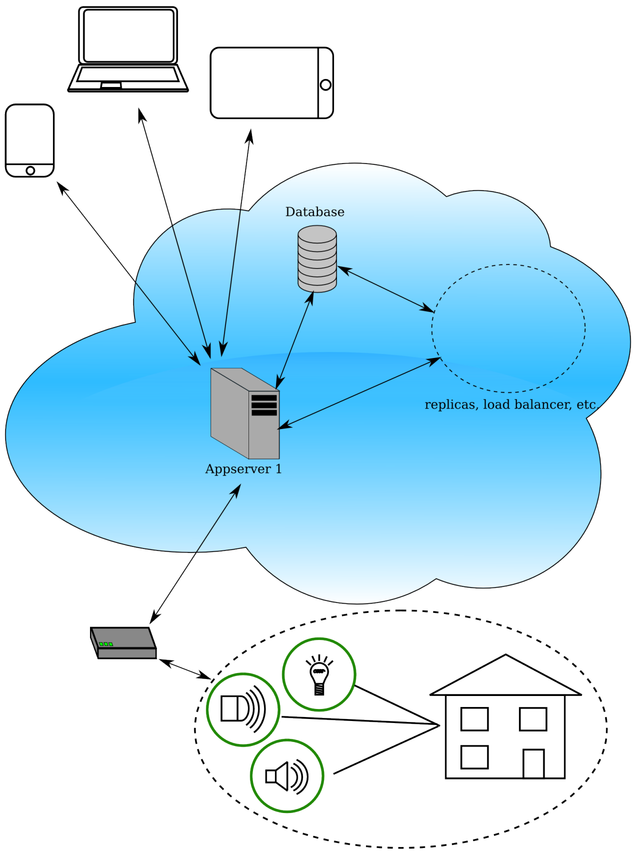


Figure 1.1: Basic Cloud Design

In the following sections, some of the more important concepts upon which this bachelor's thesis is based will be explained. Further on, some state-of-the-art protocols and technologies which also deal with home automation and cloud computing are analyzed and compared based on different perspectives, like the basic enabling technologies for an IP based IoT and application level protocols which can be used on top of those enabling technologies. In Chapter 3, some of the more important PaaS (2.3.2), IaaS (2.3.1) and cloud providers are introduced and compared based on their features. Chapter 4 deals with the advantages and drawbacks of moving home automation into the cloud. At the end, the reader will be presented with a case study, where a PaaS system (Appscale [47]) was extended by a custom API designed for IoT applications, capable of communicating with Internet of Things devices. The API serves as a proof-of-concept implementation to show what can be achieved with even a simple set of available options.

1.1 The Internet of Things

The Internet of Things (IoT) refers to the linking and processing of information of objects in daily life. More specifically, the IoT consists of a lot of different technologies which make it possible to access and control uniquely identifiable devices in our environment, such as sensors and actuators.

The term was first coined by Kevin Ashton in 1998, who then described it as “a standardized way for computers to understand the real world” [48]. In the past, the term primarily meant equipping everyday objects with RFID tags and similar technology. The idea was that those things should have the means to report information about themselves to other computers and embedded devices, for example frozen food packaging that contains an RFID tag which provides the microwave with all the information needed to cook the food. This technology has been adopted to this date primarily by companies which have to deal with a lot of logistics in order to track their shipments and products. Used in everyday objects, this technology alone might have significant impact on society by providing large amounts of additional information that could potentially be used to deal with a lot of problems that concern us today, such as waste reduction (for example by identifying the material of the waste via RFID) and energy consumption in general (e.g. when owning a photovoltaic collector, energy-intensive tasks can be automatically scheduled to execute when the conditions are optimal (during daytime, when the owner is at work)). But the idea of the Internet of Things has evolved since then to encompass ever more powerful network enabled embedded devices (sensors, switches, etc.) all around us/our houses that are interconnected to Wireless Sensor and Actuator Networks (WSANs), and autonomously collect data and act on changes in their environment.

Closely related, if not identical, to the field of the Internet of Things is Ubiquitous or Pervasive Computing, where the basic concept is that computers and small autonomous embedded devices are integrated into our everyday lives, up to a point where we are surrounded by a multitude of computing devices that make our lives easier and more comfortable.

Also related to the Internet of Things (especially the Web of Things) is the so called Semantic Web, which aims at providing machine readable semantic context to the data on the Web, so that machines can process the information content of this data more efficiently and thereby derive more information on their own without human input.

There are a lot of different areas of application for an Internet of Things, the more important concepts are introduced in the following section.

Industrial Automation and SCADA

A very important aspect of the IoT is the automation and monitoring of industrial processes. SCADA (Supervisory Control And Data Acquisition) is such an automation system. It deals with large scale industrial processes like the management of power grids or airports. SCADA systems include sensors and actuators in the industrial environment, human-machine interfaces to interact with the processes, and computer systems to automatically regulate processes. All of this infrastructure is connected to each other via different technologies. Potential technologies and concepts for this communication include M2M and WSANs.

M2M is probably the broadest term, describing techology which is capable of communicating with other devices over wired or wireless channels, which can then process and act on the provided data. In such systems, humans often act only as recipients of the data but do not actively initiate events [34]. The term can also be found in areas which use cellular networking (or satellite communication) technologies to exchange telemetry data between a large number of nodes of different kinds.

Wireless Sensor Networks consist of wireless nodes which communicate data via possibly other nodes to so called *sinks*. *Sinks* are special nodes where the data gets further processed [5]. WSNs behave similar to the Internet, in that a node does not initially know where the sinks are located, nodes forward their data to other nodes, so that that the data will eventually reach its destination. This allows the network to adjust to node failures (self-healing), automatically configure new nodes or relocated nodes (self-configuration), find the shortest route to the sinks (self-optimization) and a lot of other mechanisms that belong to those and other areas of research. The number of nodes in those networks can sometimes reach up to thousands of participants. Modern WSNs don't just contain sensors but also actuators. Such networks are called Wireless Sensor and Actuator Networks, or WSANs [48]. This technology, like so many others, was originally developed for military applications but by now has arrived in many civilian sectors, for example in industrial monitoring and environmental monitoring applications.

Automotive and Telematics

This area of the IoT deals with monitoring and regulating processes in the field of cars, vehicles and logistics. Telematics, although originally having a more general meaning, has come to be known as technology used in vehicles, such as in-car terminals for GPS, various safety mechanisms and diverse other integrated technological devices installed in today's cars [48]. This field is especially important for logistics companies, because it enables them to track their vehicle fleet in real-time and reroute certain shipments if traffic conditions are not optimal in an area. This helps the company to save money, but also reduces CO_2 emissions due to the fact that the cars aren't stuck in traffic for an extended period of time. Of course, this technology also has a lot of use cases in the private sector: cars can be tracked if stolen, security is improved by the numerous sensors (engine, tires, acceleration controllers, ...) and comfort is increased by features like remote keyless entry or automated HVAC control.

Domotics and AAL

The term domotics is composed of either *domestic robotics* or *domestic informatics*. It is a vague term that deals largely with home automation. It focuses on automating routines for the home and its owner to improve the comfort of living in an automated house. Domotics combines a lot of technological concepts in this field with the aim to automate as much of a user's life as possible and thereby improve his quality of life. As the term *domestic robotics* implies, the use of robots in a smart home to assist with certain tasks is more prevalent in this field than it is in others. Such robots can, for example be deployed to assist elderly or handicapped people with physical tasks that other simple actuators are unable to do. Thereby, they remove the dependency of those people on other humans in their daily life. This specific area of domotics is also referred to as Ambient Assisted Living (AAL) to distinguish it from the more general field of application. Also closely related to the idea of domotics is the concept of smart buildings, but this area also deals with factors that are not directly connected to humans, like reducing water and electricity consumption, reducing waste production or just generally decreasing energy consumption by intelligently using available data from around the house.

Smart Cities

Another big field of application for the Internet of Things is the monitoring of environmental processes. Such monitoring can include radioactivity levels, water quality, air quality, etc. This data can potentially be used to better understand our environment and improve our quality of life. To gather such data in large-scale environments, M2M, WSNs and other technologies are used.

Related to this field is the idea of the *smart grid*. It is one of the most important concepts of the IoT, as it deals with the growing need for power supply and stability of electrical power grids. The idea uses sensor networks to monitor nodes in the power network and perform needed tasks automatically, such as routing additional power to a node or rerouting power if a node is damaged. The demand for power varies significantly over the course of the day but the power needs for a given time are for the most part known, as a large part of the population follows a regular behaviour pattern throughout the day. This enables power suppliers to plan ahead, as is done for example with the reservoir power stations in the Austrian alps. When there is excess power (at night), water is pumped up into a reservoir and when additional energy is needed, the water in this reservoir is used to generate electricity. With the help of the smart grid, this mechanism can be scaled down to a much more fine-grained level where demand gets measured using Smart Meters, and every power supply available, including battery powered cars and other suitable battery powerd appliances, can be used to provide backup power for load spikes. This has to be coordinated with the owner's habits so as not to drain his car battery right before he wants to take a trip to the grocery store. Such a system would be much more effective than current ones, partly because the needed power does not have to travel large distances but can be used right from the neighbours car. This fine-grained behaviour also greatly increases the robustness of the whole grid, in the event of a disruption, the system can temporarily provide backup power for an area using local energy reserves and supplies.

The idea of *smart cities* / *smart planet* combines smart grid and smart buildings along with traffic control and all other smart technologies into a vision that aims at dealing with large scale problems of today's world, mainly energy consumption, by improving much of our daily lifes with the help of communication technology. By being able to place small sensors nearly everywhere, we would be able to measure certain conditions much more fine-grained than is the case in most systems today. Those fine-grained measurements would enable us to coordinate further steps for those systems much more efficiently.

With all the new technological developments, much more integrated interaction between humans and machines will be possible. The different research fields are endless, but some of the more prominent areas which the IoT has the potential to effect greatly are smart grid, environmental monitoring, healthcare, home automation, etc. There are many more, and there is virtually limitless potential for ideas in this area, which will, with ever more sophisticated and smaller devices, expand a great deal in the years to come. The main challenge for creating more acceptance among the population and thereby expanding the field of IoT will be to provide unified mechanisms and protocols which the devices can use, so that the customer does not have to deal with all sorts of different technologies.

1.2 Cloud Computing

In recent years, the term cloud computing has dominated the information industry as it got more and more popular to move resources into the so-called cloud, a transparent network of computers designed to hide unimportant information, like the specific location of files, from the users and allow them to access their resources from all over the world. Cloud computing became very popular for the end user when it got more and more common to store files in the cloud to prevent a computer fault from destroying important data, for example with the use of Dropbox, Microsoft's SkyDrive or iCloud. Google and other companies have used the basic concept of cloud computing for a long time to coordinate and compute requests made to their own sites and services, but a while ago they opened up the possibility for smaller companies and end users to write their own distributed applications and services in exchange for relatively small fees based on the number of requests to the application, the up- and download-volume and many other factors. Those offers from Google, Microsoft, Amazon, etc. made it possible for many companies to switch from a static set of Web servers serving their site/service to an elastic cloud which adapts to many circumstances in its surroundings. This system benefits both the provider and the customer, the customer because it is often much cheaper to just rent the cloud service than to build and administrate own IT infrastructure.

The main strength of cloud computing is that it can adapt dynamically to different circumstances, like the number of requests, positions of users around the globe and so on. If there are not enough resources available to handle upcoming tasks, more resources are allocated to meet the needs of the application, and if too much resources are in use compared to the actually needed amount, some of those resources are de-allocated. This reduces costs for the customer and in turn frees up resources to be used in other applications that need it more, which is advantageous for the service provider.

There are basically two deploy models for a cloud infrastructure, private and public cloud [37]. A private cloud is used only by a single company, though it can be hosted externally by a third party and still be a private cloud for that company. In the public deploy model, the access to the cloud is only possible via a public network like the Internet, as opposed to possible direct access in a private cloud. The differentiation between those two models is often not very clear as private clouds are usually also connected to the Internet to some extent.

In addition to the deploy models, there are today, three diffrent paradigms in use by cloud providers to offer their different services to customers. Those paradigms are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). From the ground up, IaaS provides (virtual) infrastructure on which a customer's system can run, PaaS provides infrastructure and environment for Web applications to run on and SaaS provides software functionality for the end-user to work with, more information can be found in Chapter 2.3.

1.3 Middleware / Gateways

A middleware is a software that unifies different technologies by offering a single standardised API to the developer [33]. Middleware has also other jobs, like hiding the distribution of a system from the developer/user. Although in the case of home and building automation, the programmer usually knows of the distributed nature of the system. The main advantage of using middleware in automation systems is that technologies from different vendors can be used as if they utilized the same technology. All the conversions to different formats, protocols and mechanisms are performed by the middleware transparent to the developer. There are essentially two ways where a middleware system can be deployed in cloud based home and building automation systems. The middleware system can be situated in the house itself, between the devices and the cloud service. In this case the cloud service does not need to handle different protocols and formats but needs only to communicate with the middleware system. If some or all of the devices that the user has in his home use non-IP based field buses, he needs such a middleware system connected to those devices to enable networking capabilities for them.

If all devices are IP-enabled, there is no inherent need for the user to install a middleware in his home, so a Web-based middleware system could be preferred. In this case the middleware is integrated into the cloud service. Different devices can communicate with the Web service using different protocols and formats, the conversion into a common format is done by the Web service itself.

Of course those two approaches can be combined if the user has both kinds of devices, IP-enabled and non-IP-enabled. The user can then use a middleware for the non-IP-based part of his system and connect the rest directly to the network.

Cloud computing, middlewares and the various IoT technologies play an essential role in the future of home automation. The user's devices will be connected directly or indirectly via a middleware to the cloud where they exchange data and receive instructions, either automated based on some defined rules, or directly from the user via a Web interface.

CHAPTER 2

State of the Art Concepts and Technologies

2.1 IoT Enabling Technologies

2.1.1 IPv6

IPv6 is the successor of the commonly used Internet Protocol version 4 (IPv4). The main reason why IPv6 is so important for an IP based Internet of Things is its large address space which uses 128 bits for the addressing of network interfaces (devices) as opposed to 32 bits in IPv4. With the already short supply of IPv4 addresses in today's Internet, IPv4 could not possibly provide unique addresses for every IoT device in the future, which means that those devices would have to be "NAT-ed" in order to connect them to the Internet. IPv6 solves this address-problem. Individual devices, ranging from a light bulb to a coffee maker would be able to connect directly to the Internet and exchange data with cloud servers.

2.1.2 6LoWPAN

6LoWPAN, which stands for IPv6 over Low power Wireless Personal Area Networks, is an effort to optimize power and resource usage for constrained devices while using the IPv6 protocol. The specification for the protocol can be found at [20]. As the name suggests, the protocol enables small wireless devices to use IPv6 when communicating over an IEEE 802.15.4 wireless network. This IEEE standard is initally not compatible with IPv6 as Maximum Transmission Unit and various other characteristics, like address resolution, differ between the two protocols. 6LoWPAN maps the IPv6 packets to IEEE 802.15.4 packets, therefore providing a sort of tunneling protocol that can be used in an environment where IPv6 is desired for small low-power wireless devices. Providing IPv6 for such devices is getting more and more important as the idea of IoT continually merges with cloud computing.

2.1.3 CoAP

The Constrained Application Protocol is designed especially for small, low-power devices. It is a RESTful protocol and can be easily transformed to HTTP, allowing enanced interoperability between devices and Web-based services. CoAP supports the basic request types GET, PUT and POST, as well as an OBSERVE type to register a client to receive push notifications when resources are modified. In IP based networks, CoAP utilizes the UDP transport layer protocol to transport data between devices. As UDP is connectionless, CoAP facilitates its own mechanisms to be able to exchange data in a reliable way. CoAP also has multicast capabilities, which further increase its usability for constrained environments by reducing bandwidth utilization for certain types of operations. The standardization for this protocol was and is still done by the Internet Engineering Task Force (IETF) CoRE Working Group [19].

2.1.4 Message Queuing Telemetry Transport (MQTT)

MQTT is a publish/subscribe protocol designed for constrained devices. It uses IP networks to communicate with those devices. The specification allows clients to register for one or more topics in which they are interested at a central server. If another client publishes information for a specific topic, all clients that registered for that topic get notified. The standard specifies various message types and message formatting but does not enforce a specific application protocol to be used. Additional information can be found at [18].

2.2 IoT Application Protocols

2.2.1 oBIX

oBIX is an open XML standard especially developed for the IoT environment. It is maintained and extended by the Organization for the Advancement of Structured Information Standards (OASIS) [39]. The basic idea behind oBIX is that each device in an automation environment can be encapsulated into an object and identified by a unique URI. Properties of those objects, like data points, are represented by so-called “facets” (the attributes of the objects), which for example represent the value of a data point, or by child objects. The use of child objects creates an object hierarchy for the whole automation system where devices can easily be discovered by traversing the object graph. Types of objects are represented by so-called contracts, which specify the internal structure of an object. Contracts themselves are basically templates for the corresponding objects, they are also specified using oBIX syntax. Contracts can contain default values which enables them to be used without further modification. From a Java perspective, contracts are like interfaces which can contain default values, so they can be used on their own without concrete implementations. Contracts can also be inherited by sub-objects which creates a hierarchy like a type hierarchy in object oriented languages. An example for a contract is shown below.

Listing 2.1: Contract example

```
<obj href="/def/Coordinate">
```

```
        <real name="lat"/>
        <real name="long"/>
</obj>
```

Listing 2.2: Concrete object example

```
<obj is="Coordinate" href="/GPSDevice/coordinate">
        <real name="lat" val="2.71828"/>
        <real name="long" val="3.141592/>
</obj>
```

This contract example shows how a `Coordinate` object is structured, it contains two data points which hold real values representing the latitude and the longitude of the coordinate. The example of the concrete object below shows how that contract is specified and implemented. If `lat` or `long` are not set explicitly, they will default to the default value of a real data point, which is zero. Objects show their contract either implicitly via their tag name (for example the `real` tag) or explicitly by having their `is` attribute set to a specific contract URI. This definition of object types allows vendors to construct their own types but does not force clients to have a language specific construct for that object type. They can treat unknown types just as normal objects which may contain child objects. oBIX also allows the remote invocation of operations via the `op` contract. Operations are invoked by sending a `POST` request with the URI of the operation and an optional input argument to the server which will return the result of the operation as specified in its contract. This allows oBIX to specify a lot of additional functionality by default, for instance, the concepts of watches, histories and alarms. Watches can be created at an oBIX server and are used to record the changes made to specified objects. When polling the contents of the watch, the client will only receive modfications made to the objects monitored by the watch since its last poll. Histories enable the server to record changes to objects and store the whole change history. They can be queried by certain parameters like the time interval of interest. Alarms are used to issue warnings when certain objects enter a state that was specified beforehand to trigger the alarm. When an alarm is triggered, the server may take steps to notify a user who deals with the abnormal situation.

The fact that oBIX is an XML definition raises the concern that it contains a lot of overhead data when transferring object data between small power- and memory-restriced devices. While those concerns are partially valid, more powerful and efficient devices are developed rapidly. Furthermore, oBIX also specifies a binary representation for the XML structure which can be packed and transferred much more efficiently and the general structure of the objects does not get lost in the process, making it also suitable for very restricted environments.

2.2.2 Sensor Web Enablement

Sensor Web Enablement (SWE) [7] describes a series of standards and encodings that can be used when dealing with sensor networks [43]. It also describes standards for Web interfaces that are conceptually situated between the actual sensors and the user and can, for example, be used to filter information. The standards were mostly designed for larger, scientific sensor networks and so do not necessarily perform well in a home automation environment. They define a great deal of detail for the description of sensor networks themselves and the information flow therein. In contrast to oBIX, those definitions provide a lot more semantic context to an entity processing the data, thus enabling it to make more informed decisions. But the semantic overhead also puts more load on small constrained devices which might not be able to handle all the information. Additionally, all this information will probably not be needed in a home automation environment because locations, capabilites, etc. are known to the user beforehand and so do not need to be communicated to the user. The most important standards in this suite for communicating sensor data over a network are the Sensor Model Language (SensorML) and the Observations and Measurements (O&M) standard.

SensorML [44] specifies models to describe the sensors themselves, using a wide range of attributes like GPS coordinates, capabilities, characteristics and also processes for a sensor. This makes it possible to describe many kinds of sensors by describing their atomic attributes. But it does not only describe the model for the sensor layout, but also, like mentioned above, things like the different processes that a sensor can execute. All those properties of sensors can be described in great detail by using all kinds of meta-information like the resolution of mesurements or the conditions under which certain measurements can be performed. The specification also supports additional operations like discovery of sensors based on criteria like the capabilities or processes of sensors and things like alarming to notify entities about certain state changes.

The Observations and Measurements standard [38] defines how the results of executed processes on the sensor-side are represented and encoded for use by an external entity. The model contains for example information about the observed property, like resolution, or the time that the result was produced.

2.2.3 IPSO Application Profile

The IPSO Application Profile [45] describes an information hierarchy for small embedded devices. An object's location is defined by its URI on a node. Data belonging to the object is also accessed via URIs relative to the object's URI. Discovery of devices on a node is supported via the CoRE Link Format. The framework also defines resource types for objects, such as *Light Control* (ipso.lt) or *Sensors* (ipso.sen), data types like boolean and integer, and other relevant information which can be used when querying data from a device. It is designed for RESTful environments and can officially be used in conjuntion with HTTP and CoAP, making it suitable for Web (and cloud) applications in conjunction with embedded devices. When using CoAP, the OBSERVE mechanism can be utilized to receive an immediate notification of an object's state change. Supported format types include plaintext and SenML [28]. Aside from the specified attributes, additional data can be added to objects on demand. This mechanism allows for the extension of objects in certain cases where the standard object data does not meet the needs of the application. The framework explicitly supports alarming, but has no built-in mechanisms to store history data of objects.

2.2.4 OPC Unified Architecture

OPC UA [13] is a new effort by the OPC Foundation to create specifications for interconnecting IoT devices. The standards contain details for discovery of devices, an information model, security, alarming, history data and much more. The address space of OPC UA is structured hierarchially but allows for references between nodes so as to represent relations. Nodes belong to classes which define their meaning and contain attributes like the name or the ID of the node. Specific node classes have specific attributes which further describe the nodes. The security model specifies different levels of encryption that can be used, based on the environment where the system is deployed. Higher security uses more resources and may therefore be unsuitable in constrained or real-time environments. The specification describes three so-called profiles which essentially represent different efficiency levels for data acquisition. In the first profile, called *Binary Profile*, data is encoded in a binary format and transmitted via TCP. This profile consumes the least resources and can be used in conjunction with small embedded devices. In the next profile, the *Hybrid Profile*, data is encoded in binary, but transmitted via the HTTP(S) protocol. This profile is especially useful for Web applications in conjunction with constrained devices. The third and most resource consuming profile is the *Web Service Profile* where data is encoded in XML format and exchanged via SOAP and HTTP(S) [14].

2.2.5 Other research in this field

An extensive analysis in the field of sensor networks and related semantics of sensor data was conducted from 2009 to 2011 by the W3C Semantic Sensor Network Incubator Group (W3CSSN-XG) [15]. In their final report, many questions regarding the structure and design considerations for implementing a standard for the representation of sensor networks are discussed and answered. On the basis of previously existing specifications, the group designed an ontology which is capable of representing sensor data and interactions.

Although JSON (Javascript Object Notation) is often used in cloud based IoT systems, no concrete specification yet exists describing a standardized format for IoT communication. JSON is in some ways more promising than XML for IoT environments. It does not have as much overhead as XML, but this means it sacrifices some semantic context and readability in favor of simplicity. Objects in JSON are composed of comma separated attribute-value pairs enclosed in curly brackets (like Javascript objects), making them highly dynamic and very simple to understand and construct. Like XML, JSON is already an integral part of Web-based services, therefore no additional adjustment is needed to process JSON objects.

2.2.6 Comparison

In the following table, application protocols are compared based on three features of their specification. *Descr. Format* indicates whether the specification for the protocol describes a format for the payload, *Descr. IF Structure* indicates if the specification describes interface structures for communicating via the protocol and *Supp. Formats* lists the transport formats which are supported.

Table 2.1: General Feature Comparison of Application Protocols

Protocol	Descr. Format	Descr. IF Structure	Supp. Format(s)
oBIX	yes	no	XML
SWE	yes	yes	XML
IPSO AP	yes	yes	plaintext, SenML[s]
OPC UA	yes	yes	binary, XML

[s] see [28]

2.3 Cloud Computing Paradigms

2.3.1 Infrastructure as a Service (IaaS)

IaaS refers to the concept of offering computing infrastructure (real computers or often virtual machines managed by a hypervisor), and often some other features like a load balancer, as trading goods. The provider leases this infrastructure to customers who pay for consumed resources (procecssor time, network traffic, etc.). In the case of virtual machines, this model often allows the customer to upload and use their own system image to the cloud where it is replicated automatically by the underlying system. The virtual machine image has to be compatible with the IaaS system for it to be deployed. Vendors will usually provide some basic images to modify as needed. The customer can use all the tools and technologies that he wants in his own virtual machine and has therefore more possibilites available compared to PaaS systems, but he also has more configuration effort.

2.3.2 Platform as a Service (PaaS)

A very important aspect of cloud computing is the concept of Platform as a Service, or PaaS. It is the idea that all the hardware and software needed for deploying a Web application to the cloud is provided to the developer by the company offering the PaaS. This means that the Web developer has to care only about the functionality of his application. When developing, he can use the API provided by the company to access important features of the PaaS, like performing database operations, or managing users of his application. As opposed to IaaS, the customer has no direct control over the underlying system, he can not add more tools or modify the system in any way that is not desired by the provider.

A benefit for the customer is that he does not have to care about patching or maintaining the underlying system, the provider will do that for him. Another big advantage of such a system is that developers have a common platform to develop for. It is easier and cheaper to use that system in comparison to creating your own cloud as a company. Typically, every PaaS provider has its own system and its own API. This of course makes it difficult to switch providers, but some newer systems try to provide compatibility with existing technology. For example, Appscale is in large parts compatible with Google App Engine applications. This compatibility of course attracts more developers to such a system, as some of them already have the required know-how to build applications for it.

2.3.3 Software as a Service (SaaS)

Software as a Service is a cloud computing model where specific software is provided to the user via a Web interface/site. The user of the software does not have to worry about installing and running the software, he connects to the service via a client (Web browser) and can use it. More prevalent examples for this kind of model are Google Docs [25] as an online office application or Gmail [22] for managing emails online.

2.3.4 Comparison

The following table shows advantages and drawbacks of the individual paradigms. While higher level systems will usually have less administration overhead, they will also offer less freedom to the developer/user. *Access level* indicates the level on which the developer can interact with the system, *Maintenance* shows the scope in which the developer himself has to conduct maintenance and *Avail. Features* shows which set of features is generally available at this level.

Table 2.2: Comparison of Cloud Paradigms

Paradigm	Access level	Maintenance	Avail. Features
IaaS	OS	full	full
PaaS	deploy[1]	application	APIs
SaaS	application	none	application

[1] Developer can deploy and manage his application

CHAPTER 3

Cloud Technology Survey

3.1 PaaS Providers

The next section contains a short overview of some of the more prominent PaaS systems in today's market. Listed are commercial systems as well as open source projects, which can be used to build a private cloud PaaS infrastructure. In the feature comparison (Section 3.1.7), some of the features of the PaaS systems are listed, including an overview of the supported languages of each system. As none of the PaaS systems listed has the ability to communicate efficiently with home automation technology (only HTTP and similar), this aspect is omitted in the feature comparison at the end of the section.

3.1.1 Google App Engine

GAE [23] is a PaaS cloud service started by Google in 2008. Users can deploy Web applications in several languages including Python, Java, PHP and Go. Due to the Java support, several other JVM languages can be used with the service. The service is free up to a certain resource usage, after which the customer will be billed based on his excess resource usage, including CPU time, bandwidth, memory, etc. Google offers several APIs for developers developing Apps for GAE which utilize the underlying Google services. For example, the Memchache API, which makes it possible to store key value pairs across requests for faster access.

3.1.2 Windows Azure Web Sites

Windows Azure Web Sites [9] is a part of the Windows Azure cloud platform. As opposed to the Windows Azure Virtual Machines part, Web Sites focuses on the deployment of Web applications. Similar to GAE, Web Sites offers a free tier with restrictions to resource usage.

3.1.3 AWS Elastic Bean Stalk

The Elastic Bean Stalk [3] is part of the Amazon Web services and focuses on the deployment of Web sites. Like GAE and WA WS, there is a free tier and a pricing model for resource usage that goes beyond the free tier limit. In contrast to most other PaaS providers, it is also possible to choose whether to use a linux or Windows host OS for the Web application.

3.1.4 Heroku

Heroku [17] is a service that started out hosting Ruby applications but grew substantially over the past years and now supports multiple programming languages. Like other providers, it offers a free tier with restricted resources.

3.1.5 Appscale

Appscale [47] is an open source PaaS that can be deployed on multiple infrastructures. Its main feature is its compatibility with GAE Web Applications. Existing GAE Applications can be deployed to an Appscale installment without modification. To achieve this, the project uses modified Google APIs. Up to now, Appscale supports Python and Java as programming languages. The system can be run as a cluster with pre-defined nodes, or in a cloud environment where it will spawn and destroy nodes based on resource usage. Appscale itself is only a PaaS layer which needs an additional infrastructure component to run on.

3.1.6 Cloud Foundry

Like Appscale, the Cloud Foundry [41] project itself is just a PaaS layer which needs additional infrastructure, but the initial developer and provider Pivotal also offers a complete cloud service like other commercial providers. Cloud Foundry is open source under an Apache 2.0 license. It supports multiple programming languages and infrastructure components (e.g. Amazon EC2, CloudStack).

3.1.7 Feature Comparison

The features listed in the table below are:

- **Open Source**: Indicates whether the system is available as open source software, *partially* denotes that at least a part of the software is available under an open source licence.
- **Full IPv6**: Indicates if the online resources of the provider are fully accessible via the IPv6 protocol. If the system is purely open source, this aspect does not have significance as it is up to the user to install it in an IPv6 enabled environment.
- **Supp. languages**: Shows the number of programming languages that can be used to develop for this system. The next table lists the available programming languages in detail.

- **Private Cloud**: Indicates whether the system can be used to set up a private cloud, only systems that can be installed in a non-proprietary environment can be used in such a way.
- **Supp. Infrastructure**: Indicates on which type of infrastructure the system is able to run. *Proprietary* denotes that the system is run on a proprietary infrastructure used by the provider.

Table 3.1: General Feature Comparison of PaaS Providers

Provider	Open Source	Full IPv6	Supp. languages	Private Cloud	Supp. Infrastructures
GAE	partially	yes	4	no	proprietary
WA WS	partially	no	4	no	proprietary
AWS EBS	no	no	6	no	proprietary
Heroku	no	no	5	no	proprietary
Appscale	yes	-	2	yes	many[A]
Cloud Foundry	yes	-	3	yes	few[CF]

[A] VirtualBox, Amazon EC2, Google Compute Engine, Rackspace, Eucalyptus, Xen KVM, OpenStack, CloudStack; as cluster on arbitrary VM
[CF] OpenStack, vmware, Amazon WS

Table 3.2: Programming Language Support of PaaS Providers

Provider	Python	Java	PHP	Node.js	Ruby	.NET	Other
GAE	●	●	●				Go
WA WS	●		●	●		●	-
AWS EBS	●	●	●	●	●	●	-
Heroku	●	●	●	●	●		-
Appscale	●	●					-
Cloud Foundry		●		●	●		-

3.2 IaaS Providers

In this section, some of the more popular IaaS providers are presented to the reader. The list contains commercial IaaS systems provided by estblished cloud computing companies such as Google and Amazon as well as open source systems like Apache Cloudstack. Located at the end of the section is a table showing the more interesting features of each system.

3.2.1 Google Compute Engine

As the IaaS branch of the Google services, Google CE [24] deals with providing bare infrastructure to customers. The system is able to host Debian and CentOS operating systems and automatically scales them based on resource usage. As with most commercial offers, the customer can decide the geographical region in which his data should be hosted.

3.2.2 Amazon Elastic Compute Cloud

As one of the fist commercial IaaS providers, Amazon has always continued to develop its cloud computing services. The Elastic Compute Cloud [4] supports a wide range of operating systems including Windows Server and freeBSD.

3.2.3 Windows Azure Virtual Machines

Virtual Machines [8] is Microsoft's offer in terms of IaaS services. It features a wide range of supported operating systems and other services like automatic scaling and geographic positioning based on user preference.

3.2.4 CloudStack

Apache Cloudstack [11] is an open source (Apache License 2) IaaS system developed by Cloud.com. It supports multiple hypervisors and thus enables the use of a lot of different operating systems based on the hypervisor used.

3.2.5 OpenStack

OpenStack [40] is an open source IaaS project currently managed by the OpenStack Foundation. Like CloudStack, it supports multiple hypervisors and can therefore host a multitude of operating systems in the cloud environment. An important feature of OpenStack is its compatibility with Amazon EC2 and Amazon S3 APIs. Due to this compatibility, applications created for those Amazon services can easily be ported to an OpenStack environment.

3.2.6 Feature Comparison

Table 3.3: Feature Comparison of IaaS Providers

Provider	Geo. Location[1]	Supp. OS[2]	Automatic Scaling	Private Clouds
Google CE	yes	Debian, CentOS	yes	no
Amazon EC2	yes	var. Linux, freeBSD, WS[WS]	yes	no
WA VM	yes	var. Linux, WS	yes	no
CloudStack	-	various[3]	yes	yes
OpenStack	-	various[3]	yes	yes

[1] Users can adjust geographical location of instances
[2] Operating systems supported by the system
[3] Depends on used Hypervisor
[WS] Windows Server

3.3 IoT Cloud Service Providers

3.3.1 Xively

Xively [27] is a cloud service that can be used to monitor and process home automation device data. It enables the user to push data from his own sensors into the cloud and to process that data using the mechanisms provided by the site. The user can decide for himself if his sensor data should be public or private and configure his data streams accordingly. The service is free of charge for developers but has some minor drawbacks like a limited amount of API calls per minute. It supports multiple data formats for input and output, like XML, JSON, CSV and EEML. A lot of different libraries are available for diverse platforms to provide easy access to the features provided by Xively. Up to May 2012, Xively was known as Pachube and received a lot of medial attention when it was used by the Japanese public to collectively monitor radioactivity levels around the area of Fukushima after the nuclear desaster in March 2011.

3.3.2 SicsthSense

SicsthSense [46] is a service that allows the user to record sensor data in the cloud, either by regularly polling the resource or by means of pushing the data from the sensor into the cloud system. The user can create public or private streams that represent a series of sensor recordings which can then be viewed on the Web site and used through the provided API. The system supports the HTTP and the CoAP protocol for polling devices. Pushing is only possible by using HTTP. The engine behind the Web site is available on github under a modified BSD license[1].

3.3.3 Carriots

Carriots [6] is an IoT cloud service which lets the user post data from his devices to datastreams in the cloud by using the REST API provided by the system. The service provides some interesting features when it comes to data proccessing. Users can, for example, put triggers on various stages of the data processing cycle, from receiving the data to persisting it into the database. These triggers can be used to execute actions that should be performed with the data, like posting it on a twitter feed. Triggers and other actions can be easily configured in the provided Web GUI.

3.3.4 Other Research in this Field

Various research teams have developed ideas and implementations of cloud services to support their research in specific areas of interest within the IoT, some of those ideas and concrete implementations will be presented in the following section.

In their research paper about *Cloud Controlled Intrusion Detection and Burglary Prevention Stratagems in home automation Systems* [36], Anindya Maiti and Samba Sivanesan proposed and implemented a cloud service capable of detecting burglars and notifying law enforcement

[1]`https://github.com/sics-iot/sicsthsense`

agencies if requested. The system uses infrared security cameras in each room to be able to detect motion events. If such an event is registered, the home automation system notifies the cloud system which in turn notifies the home owner of a possible intrusion. The owner can be notified via SMS, e-mail or other efficient means. After the home owner is notified, he can decide to watch the camera feeds from his home via the cloud service. If he finds that there is some intruder in his house, he confirms the alarm and the cloud system takes steps to counteract the intrusion (e.g. notifying law enforcment, ringing alarm bell).

In [16], a cloud service is proposed that can be used in the automotive domain. Vehicular data is gathered and processed in real time. This data is used to provide improved safety and co-ordination between road users as well as real time data such as parking information or traffic congestions.

A paper by Charalampos Doukas and Ilias Maglogiannis [10] focuses on the healthcare aspect of ubiquitous and cloud computing. In their paper they propose and implement a cloud based system that is able to monitor a user's vital signs with the use of various embedded sensors. The system utilizes REST based communication and is highly scalable.

3.3.5 Feature Comparison

Table 3.4: Feature Support of IoT Cloud Providers

Provider	JSON	XML	CSV	HTTP(S)	CoAP	Pull[1]	Push[2]
Xively	●	●	●	●			●
SicsthSense	●			●	●	●	●
Carriots	●			●			●

[1] The cloud provider can pull data from an online resource
[2] Data can be pushed from a resource to the cloud provider

3.4 Conclusion

PaaS systems allow for an easy and fast development of Web applications but they lack important functionality which is needed for home automation and an Internet of Things in general. Important features to enable this technology would include the ability to talk directly to individual devices via lightweight protocols such as CoAP and to regularly monitor devices in the background. Some cloud systems provide SaaS in conjunction with home automation, but the user is bound to the specific service and no new functionality in the form of applications can be developed. An elegant solution to this problem would be for PaaS providers to equip their systems with basic IoT capabilities and provide developers with an API to interact with this functionality. This would enable the developers to create IoT applications in a quick and easy way and thereby increase the range of available services.

CHAPTER 4

Cloud Systems for Home Automation

4.1 Functionality of Home Automation in the Cloud

The basic idea behind moving home automation into the cloud is to provide as much convenience for customers as possible. In the ideal cloud application, the customer has only to install a home automation device in his or her house and perhaps register it with the cloud application. Any further steps would be an annoyance for the customer. It should also be completely irrelevant which manufacturer produced the new device. There are of course some prerequisites that must be met for such a system to be able to use the cloud. For systems using a fieldbus to communicate, it is necessary to have a gateway between the bus and the Internet. If IPv4 is used as the Internet layer protocol, it is also impossible for the cloud application to contact each device with its own address, whether it is because NAT is employed or just because there are not enough IPv4 addresses for each home to connect possibly dozens of devices directly to the Internet. IPv6 makes such a system much more feasible. The great benefit of using cloud computing for home automation is that as long as a device can be contacted via the Internet, the user does not have to care about which specific proprietary (or open) protocol it uses. The middleware of the cloud application should handle exchanging data with the device for him and present the information to him in a standardised way. Basic use cases for a cloud service include:

- Data logging and processing in the cloud.
- Monitoring and controlling of devices via the service.
- Automatic regulation based on some set of rules which have been defined and are processed in the cloud application.

4.2 Benefits of Cloud Computing

4.2.1 Convenience

Cloud computing provides a lot of features to the user which mostly increases convenience when dealing with home automation. The number of people owning smartphones or other related mobile devices increases steadily. In coming years, most of the population will be connected to the Internet at nearly all times. The combination of cloud computing and mobile devices like smartphones and tablets enable easy and convenient ways to remotely control home automation systems.

Some features and possibilities listed below assume an open cloud Platform as a Service with an API available to Web developers but most of the features will also be available in a closed commercial or a private cloud system that provides means for home automation.

- **Worldwide availability of the data**: Cloud providers will usually have a much higher reliability and availability than private Web servers. The provider might also have certain SLAs stating the capabilities and limits of the service so that a customer can assess his risks based on these data.
- **Cost reduction for the customer**: No permanently online Web server is required. This means no more maintenance costs for a privately owned server. The cloud provider will usually bill the customer for computing time and bandwidth but often those costs are less compared to the potential risks of running the system on your own.
- **Highly reduced risk of losing the data**: Storage is replicated in the cloud.
- **Highly reduced risk of server outage**: Cloud computing is highly distributed.
- **Cloud platforms support multiple protocols**: To talk to automation devices, the user does not have to stick to a certain vendor. Of course not all cloud platforms support every proprietary protocols and formats, but the user/customer can choose from a wider range of devices and the provider might integrate more functionality (protocols/formats) over time as the platform expands.
- **Cloud platforms offer device data in a uniform way**: This essentially means that a middleware is integrated directly into the cloud system to offer a common API to the Web application developer. This has the advantage that the user does not have to care about different vendors himself but is presented with a consistent interface to interact with his devices.
- **Easy and efficient**: A cloud platform can offer easy and efficient ways to interact with devices and automate certain routines for the user.

4.2.2 Advanced Use Cases

With the increased availability and the potentially massive processing power of the cloud, things like face and speech recognition, burglar detection, automatic alarming and many other features can be made available to the customer. Examples of such use cases are:

- **Automatic Burglar detection**: Based on the location of the legitimate users (or some other factor, like the current time), the system can potentially detect intruders and inform the home owner. The home owner can then take steps to counter the intrusion (possibly by using the system itself [36]).
- **Automatically inform emergency services**: Emergency services like police, fire brigade or even emergency medical services can be notified automatically based on the available data. Video cameras and motion detectors can be used to detect intruders, smoke/heat detectors can be used to alert the fire brigade and wearable electronics can monitor vital signs and notify an ambulance in case of an emergency.
- **User recognition**: Users can be recognised based on their appearance or their voice, this can be used by the system to offer individualised services/routines for each user. An example would be to play specific music based on the recognised person.

4.3 Risks and Drawbacks of the Cloud

4.3.1 Effort and Costs

Regardless of whether a customer uses a commercial cloud service on the Internet or he builds his own private cloud system, there are some effort and cost connected with using such systems.

When building a private cloud platform for a building/company (be it a residential home or a commercial building), there are quite a few things one should account for. First, there are the costs for acquiring and configuring the hardware, whether the cloud platform is virtualized or not, at least some hardware infrastructure is needed to get the system going. Depending on the size and throughput required, the needed infrastructure can range from a simple PC to a full blown server-cluster. Aside from that, the system needs to be configured and maintained, if it incorporates crucial functionality for the automation network 24/7 attendance of a human controller may be necessary in case of a malfunction. All of this combined can have quite a financial impact on a small business and may therfore not be a constructive idea.

Using a commercial cloud system can be a better option for smaller companies. In such systems, the provider takes care of all the maintenance work necessary. Depending on if the customer is using an IaaS or a PaaS system, he will only need to patch the system or the application himself from time to time. This business model takes the maintenance cost of the hardware infrastructure away from the customer but introduces another cost factor, the cloud provider. Usually, every cloud provider will bill a customer based on the utilization of resources in the cloud system, but this is likely less than the cost of running and maintaining the system in a private setting.

4.3.2 Privacy and Security

Of course there are severe security concerns involved when thinking about sending all the data of your home to the servers of some company, those can largely be classified into two types, security concerns and privacy concerns. Those two types have a lot in common but the results for a home owner are somewhat different.

Privacy concerns deal with the fact that by uploading all the information about your surroundings, your cloud service provider can potentially analyze all this data and your life becomes transparent to the company, which is able to make a profit from your behaviour by either selling your profiling information to third parties, or just use it for their own marketing strategies. The information that would be given away to those companies in the course of a day include favorite television or radio channels, favorite food and drink (if it is RFID tagged), movement data based on all the sensors in the house, electricity and water usage, and much much more. By also integrating personal electronic devices like smartphones into the Internet of Things, a person's whole life could potentially be monitored and analyzed. All this information is very valuable for advertising companies which will then serve a user with personalized advertisements based on his behaviour throughout the day [35]. All this does not directly harm the user, but it is clearly a scary thought to consider. With all the news about global surveillance by all sorts of intelligence agencies, this thought can be taken one step further. Think of a world where everybody has a multitude of home automation devices connected to the Internet. In such a world, intelligence services could potentially monitor every step a person makes, create behavioural patterns and see if this individual is behaving normally or is exhibiting a strange behaviour, which could indicate a potential threat.

Security concerns deal primarily with the fact that the information that gets sent from a home to the cloud could potentially be used to harm a user in some physical or financial way. Even if the company whose Web service he/she uses is completely trustworthy, there is no guarantee that the Web service, let alone the communication between the house and the Web service is completely secure [32]. A hacker could potentially hack the database of the service or just sniff all the data transferred between the house and the service. This data could contain hints about the user's current location (even his current position in the house based on movement sensors usually used for smart lighting) or reveal the presence and/or positions of security systems. With all the information from an automation system, a burglar could know when a home owner is away on a trip and can then evade all the security systems to break into his home. This of course is much harder for the criminal if the traffic is encrypted properly and the Web service uses state-of-the-art encryption techniques to securely store the data, but nevertheless the possibility of intrusion can never be completely dismissed.

To give some overview of the possible attacks that can be performed on computational networks and the results for the end-user, some of the more relevant types are listed below [48][30][29]:

- Skimming: Mostly relevant for RFID and other technologies that operate with some kind of near-field electromagnetic technology to exchange or simply provide data. The attacker reads the information from the device without the user's agreement or knowledge. This attack could potentially be used to create movement patterns and profiles to analyze customer behaviour or simply to track people.

- Eavesdropping: Like described above (sniffing), the attacker might simply record data that enters and exits the network/system and thereby gain access to sensitive information that could potentially be used to harm the operator.
- Tampering: The attacker modifies data on a device, thereby changing system behaviour. This could potentially be used to fool security systems into ignoring a threat.
- Spoofing: Similar in some ways to tampering, but the attacker does not change the data on a device but pretends to be some other person/device by providing falsified information to the system.
- Killing: The attacker might be able to remotely destroy a device by exploiting a weakspot in the design (either the physical or the software design) which could potentially have catastrophic consequences, especially in industrial or health care environments.
- Denial of Service (DoS): The attacker prevents legitimate users/devices from accessing a resource by putting too much load on the resource so that legitimate users cannot get access. This attack is often done by using distributed computers in the network to be able to overwhelm a single high capacity target (DDoS).
- Jamming: Mostly relevant for wireless communication, the attacker disrupts the wireless connection. Traditionally done by using high energy radio transmitters that broadcast on the same frequency as the legitimate system and thereby overrides its lower-power signal. Some modern technologies are harder to jam because of their modulation but some are also easier to disrupt. For example, Bluetooth will not transmit on a channel that is currently in use and can be disrupted simply by occupying some or all of the used channels with a continuous signal.

There is a lot of research going on to address those security and privacy topics, some encryption standards for the IoT have been developed and implemented that can effectively prevent some of those attacks like skimming, or eavesdropping. Essentially all attacks that an attacker can use to gain information from the system or modify the system to his benefit. Other attacks simply cannot be prevented without finding and removing the source of the attack, like Denial of Service or jamming.

Another factor that is important to consider in this topic is the infrastructure of the cloud system. In a private cloud system situated in the user's home and maintained by the user himself, there will almost certainly be security loopholes that an attacker can exploit to gain access and/or harm the system. This is not because the user is too stupid to secure his system properly, but because even in an up-to-date system, there will always be security loopholes that are not yet known to the public (zero-day exploits) due to bugs in the programming or faults in the specification of some encryption algorithm. This is the case even with a well-informed user that maintains his system regularly and applies up-to-date patches, but such users will be the minority when home automation hits the broader public in a few years. For a standard user, it is arguably much more secure to use an established online service to take care of his home automation needs. This removes the need for the user to care about the security aspects of his automation system and passes the responsibility on to the service provider who generally has the means to put much more effort in keeping his system up-to-date. A big cloud provider also usually has ways to fend off disruption attacks like DDoS due to the complexity and size of such a system.

4.4 Conclusion

As is the case with a lot of online services today, the user trades privacy for increased convenience. Most of those services provide easy and convenient ways for the user to perform certain tasks, be it searching the Internet or keeping in contact with old friends. Those features are mostly free of charge, which is prerequisite in today's world to attract the most possible users. This is not inherently a bad thing, but the company needs of course some ways to make money, and so the statement "if you are not the customer, you are the product" holds for most of those services. All considered, a typical home owner living in a smart home is probably better off (when considering security) using the services of a larger cloud provider than running his own system. One could argue that a single home/person is not important enough to be hacked anyway, but such arguments will soon lose their validity when the IoT is established in the daily lifes of the general population because as the saying goes "crime never sleeps", meaning that criminals will soon adapt to this new opportunity and people need robust security solutions for their home. Of couse, as mentioned above, using a public online service for home automation leads to a whole lot of privacy concerns involving the handling of sensitive user information by the provider. Even if the data is encrypted, the provider needs the means to decrypt the data to be able to present it to the user, and thus can misuse the data for his own ends. Depending on if the service is payed by the user or not, it will have less or more incentive to misuse it, as with so much services today, the user will have to decide for himself what his privacy is worth.

CHAPTER 5

Proof of Concept PaaS for Home Automation

5.1 Overview

For the purpose of demonstrating some of the concepts mentioned in the previous chapters, a proof-of-concept was conducted with the open source PaaS Appscale [47]. Existing services like GAE, Microsoft Azure and most other PaaS providers lack important features for home automation, like the CoAP protocol, and only allow socket connections to other devices via HTTP. The idea behind this proof-of-concept was to provide an IoT API based on CoAP for home automation devices. As the Java version of the AppServers used in Appscale could be used directly in conjunction with existing code from the IoTSyS project, the Appscale project seemed the most promising for conducting this research. Figure 5.1 shows a graphical overview of the existing APIs and the new home automation API that has been added.

The basic layout for the experimental setup consists of various home automation devices which are connected to the IoTSyS gateway via diverse protocols and formats. The gateway transforms all data into a uniform oBIX representation and offers it via a CoAP server to the cloud application. Upon request by a user, the cloud application initiates a request to the gateway to read, modify or invoke an object in the system. The conceptual layout is shown in Figure 5.2.

The next sections contain a short list of basic use cases, the general working of Appscale and how the API was implemented in Appscale.

5.2 Use Cases

The basic operations of pushing data to an object, pulling object information and recording an object's history are already useful for a lot of tasks in home automation systems. Although the system does not support automated routines yet, the user can monitor and control his automation system via Appscale applications and their Web interfaces from any device which has IP/HTTP

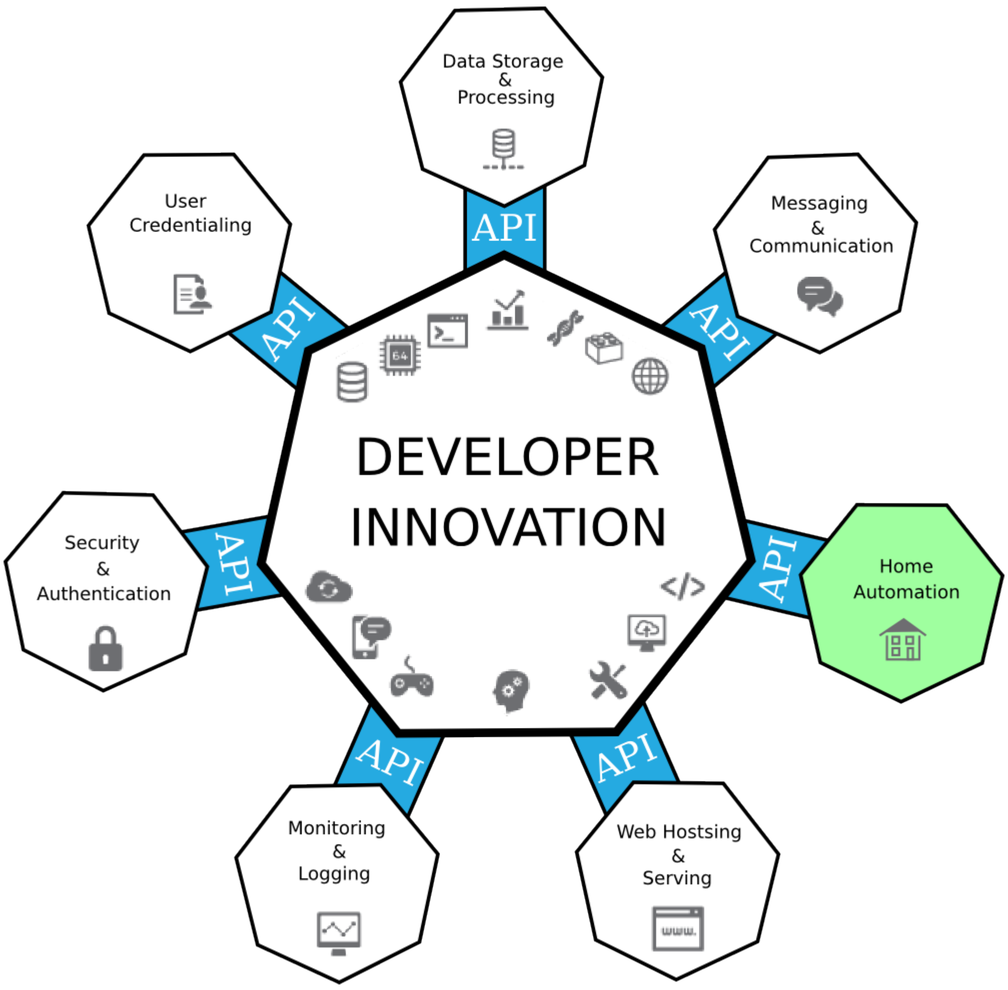


Figure 5.1: Extended Appscale API Layout

capabilities, including smartphones, tablets and all kinds of other mobile devices. Some basic use cases are listed below:

- Monitoring and controlling the home automation system when on vacation or just away on a trip. Sometimes it can be useful to have access to lighting, HVAC or sunblinds when away from home, for example, to pretend that someone is still at home so as to prevent burglars from breaking into the house or just simply to start up the heating some time before arrival to come home to a cozy house.
- Analyzing the electrical power needs throughout the day: If the user possesses a smart meter or a similar device to measure his power usage, he can use the history service of the API to record this usage. In conjunction with histories of other smart devices, he can analyse how and when he uses power and can potentially adapt his behaviour to reduce his power consumption.

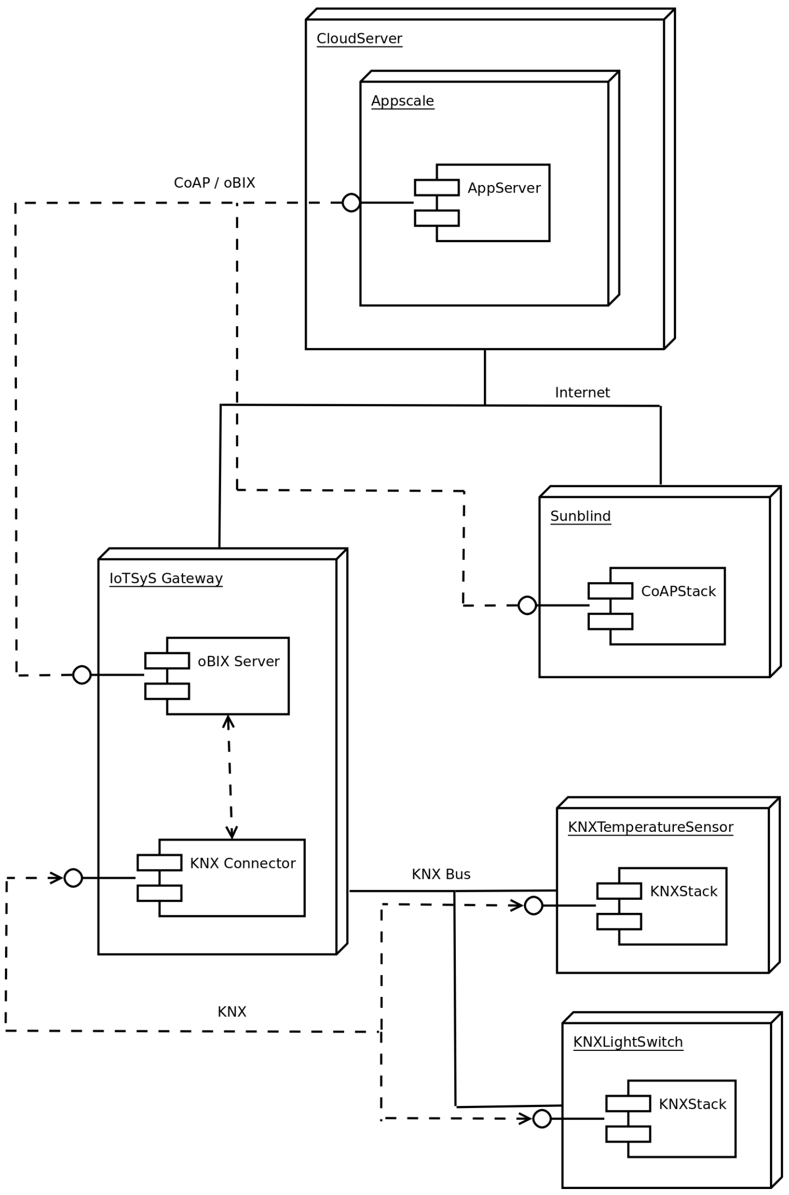


Figure 5.2: Deployment Diagram for a Potential Test Setup

5.3 Introduction to Appscale

For the work of this thesis, Appscale version 1.11.0 was used. Appscale is an open source PaaS maintained and updated by Appscale Systems. It is capable of running Google App Engine (GAE) applications and can be deployed on a minimum of one node, that is, one server or virtual machine that hosts all vital functionality for the system. The system uses a large number of different tools, technologies and programming languages to achieve its goals, for example, tools

like Apache Zookeeper [12] to keep track of services (important when scaled to more than one node), GOD [42], a ruby process management framework, to keep track of all the processes running on a node, NginX for load balancing purposes, and much more. Used programming/scripting languages include Java, Ruby, Python, Bash, etc. One of the more important technologies (for the implementation of the API) used in the system are Google Protocol Buffers [26] which are used internally to communicate between servers.
The platform uses 4 basic types of servers/nodes in the system:

- A master node which manages the system, starting up new nodes and destroying unused ones.
- App engine nodes, where each node can host several Web applications. The same application can be replicated to more than one app engine node.
- Database nodes, where exactly one node runs the datastore master and other nodes run replicas called datastore slaves.
- A Zookeeper node used to keep track of the available services in the cloud environment.

Apart from those large-grained node types, nodes have other additional functionality like load balancing, memcache service, taskqueue master and slaves and login functionality. The memcache is a reproduction of the memcache service provided in the Google App Engine where information (objects) can be stored in memory for faster access than from the database. The memcache can store information across sessions but by default gives no guarantee that objects will be available in the future. The taskqueue service is a reproduction of the Google App Engine taskqueue API where small work units can be executed outside of a user request. Appscale can either run as a cluster on a number of initially specified hosts or in cloud environments like the Amazon Elastic Compute Cloud [2] or Eucalyptus [21]. In the cluster configuration, Appscale has currently a limited range of options, supporting only one, four or eight nodes. In the cloud configuration, the system can expand or shrink freely based on current load.

The bulk of the infrastructure is written in Ruby and Python. Java is only used for the Java application server which is essentially the local testserver used for GAE applications with some modifications to the backend code by Appscale Systems. The modifications to the code include the capability for the testserver to communicate with the remote Appscale-specific API services like the database server instead of simulating the behavior of the various APIs locally. When a new Java application is deployed to Appscale, a new Java Virtual Machine is started which runs an instance of the application server, so basically every application gets its own Web server on its own port.

5.4 Implementation

5.4.1 Communication Options

The main communication options for the API are CoAP as a transport protocol and oBIX plaintext as the format of the payload. In addition to plaintext oBIX, binary oBIX and the application-link-format are also supported. Both protocols and formats are easily extendable to deal with other protocols and formats like HTTP and JSON. As oBIX is the main means of data representation between the gateway and the application server, the whole API structure is based on oBIX objects and contracts. This means that additional formats for data representation will have to use the oBIX object structure for encapsulation and representation. Currently, the three basic request types GET, PUT and POST are supported for any given protocol.

As the API functionality is integrated directly into the application server (including the recording of history data), the API does not inherently support application replication by Appscale. Manually polling and modifying objects will not have any undesirable effect as those operations do not store information on the server, but there could be complications when recording history data, see Section 5.4.6 for more details.

5.4.2 oBIX Contracts

The API focusses on the oBIX format, so objects in the API are largely based on oBIX objects, containing all important fields of those objects. Up to now, only basic oBIX contracts have been implemented as concrete objects, more involved objects are simply represented as standard objects to the Web developer. Supported object contracts include:

- Basic contracts which are represented by tags in the XML syntax, like datapoints, lists, operations.
- Watches, including the input and output object types used by the various operations.
- Histories, including the input and output object types used by the various operations.
- Feeds, mainly to enable the recording of history events via a Watch.

All those objects are represented by Java classes and are instantiated dynamically via reflection, based on the incoming object's contract.

5.4.3 Used Frameworks and Tools

To integrate the project as seemlessly as possible into the existing IoTSyS project, Gradle was used as the build automation tool, although Apache Ant was chosen to deploy the code to Appscale itself due to the fact that Ant is used in the Appscale build process and is therefore usually pre-installed on those systems. To conform to the latest version of the oBIX toolkit used in the IoTSyS project, the project `iotsys-obix` is used for compatibility. As CoAP framework, Californium [31] was used, as it is the most up-to-date implementation of the CoAP protocol stack currently available for Java.

5.4.4 Google API Structure

Every API used in the Google testserver uses the same basic class layout. Figure 5.3 displays the layout based on the iotsys-appscale-API.

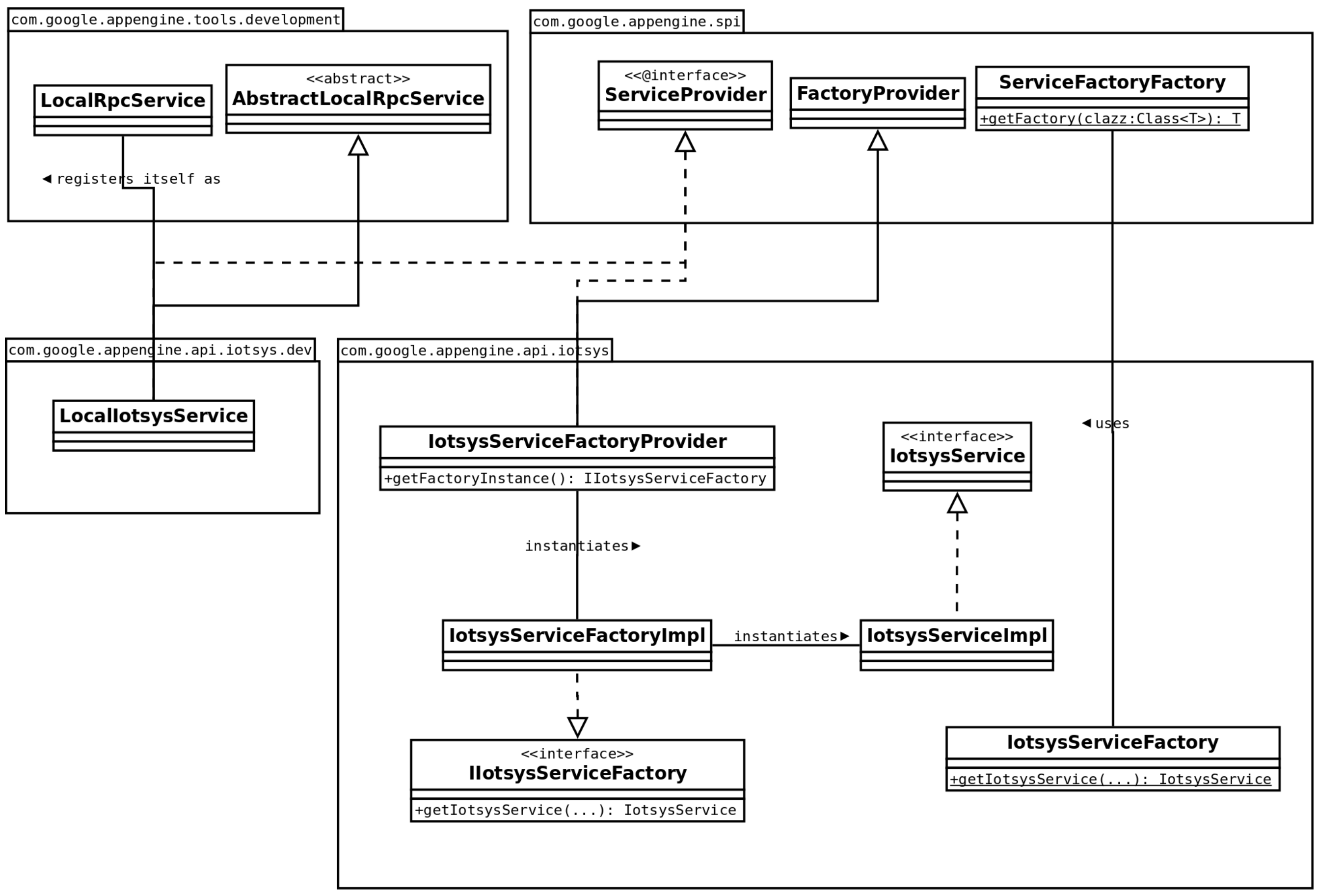


Figure 5.3: Appscale API Class Structure

The structure of an API contains:

- A class extending the `FactoryProvider` class and implementing the `ServiceProvider` interface provided by Google. By implementing the `ServiceProvider` interface, the server code can later automatically find all annotated classes of a specific type using the `java.util.ServiceLoader`. This class returns a factory able to create a service object for this API. In the case of the IoTSyS API, this is `IotsysServiceFactoryProvider`.
- An interface specifying the capabilites of the service factory of the API, in this case `IIotsysServiceFactory`.
- An implementation of the service factory interface, this class instantiates the implementation of the service (`IotsysServiceFactoryImpl`).
- An interface specifying the capabilities of the actual service of the API (`IotsysService`).
- The implementation of the service interface (`IotsysServiceImpl`).

- A class used by the developer to retrieve the service object. This is usually named [API-name]ServiceFactory, but has nothing to do with the other service factory classes. This class has static methods to get a service object by obtaining the factory provider object for the API via the Google code and using its methods to create the service instance.

- A class implementing the `ServiceProvider` interface and registering itself as a service provider of type `LocalRpcService`. This class extends `AbstractLocalRpcService` to be able to return a string identifying its package, explained in the following paragraph.

There are basically two different environments with different levels of privilege used in the application server. In the restriced environment, which is the standard for all code in Web applications, the code can not read or write files (with the exception of the application folder), can not open sockets except via the URLfetch API and is also restricted in many other things. In this environment, it is impossible to use Californium [31] (or any other similar framework for that matter) to make CoAP requests by using UDP sockets without modifying the underlying Google code. But there is another environment in which the `LocalRpcService` of each API runs. This environment can be reached from the unprivileged environment by invoking the `ApiProxy`, specifying the package to find the right `LocalRpcService`, the method to call in this class and a Protocol Buffer in byte array form, serving as input parameter for that method. The `ApiProxy` looks for the local RPC service of the given package and then calls the specified method via reflection in a privileged context. Through this process, it is possible to use Californium from the privileged environment (`LocalIotsysService`) without modifying any classes that are not part of the custom API. The process of invoking methods in the privileged environment is shown in Figure 5.5.

Protocol Buffers are a technology developed by Google that enables the fast and easy creation of custom protocols. Those protocols are designed to be lightweight and platform independent. The definition is written in an object oriented syntax and can be converted to different language constructs using the `protoc` tool provided by Google. Those Protocol Buffers (Protobufs for short) are used to uniformly convey messages from the unpriviledged environment to the priviledged code in the testserver and probably also in the GAE environment.

5.4.5 Restrictions for Cloud Applications

There are certain functional restrictions for a framework that works in a Web-based environment. For instance, the timeout of CoAP requests might be longer than the timeout of the HTTP request that triggered the CoAP request. To solve this problem, the maximum request timeout for a CoAP request was set to 20 seconds. Another problem in a Web environment is that for each CoAP request that is issued, a new port has to be opened, which can lead to a shortage of available ports if there are a lot of requests to an application. This problem could be avoided by using the same port for different requests, but the used version of the underlying framework (in this case Californium) does not support this kind of port multiplexing. As the number of ports is already limited, CoAP OBSERVE push notifications were not implemented in the API.

5.4.6 Data/History recording

As persistent history recording in the Appscale database proved to be more difficult than previously thought, history data is for now only stored in a HashMap by object location (host, port, uri). Due to the fact that the recordings exist only locally on the application server, inconsistencies with history data can occur when an application is replicated to more servers. Those inconsistencies happen when a user starts a history recording based on Watch recording (see below) on a server (server A) and logs off to check back later with the application. When the user logs on again, there is no guarantee that he will be assigned to server A again, if he is instead assigned to server B, he will not find any trace of the history that he has started on server A. If he then starts the Watch recording again, the two servers (server A and server B) will both poll the changes of the same Watch, so they both only get fragments of the whole dataset because the Watch discards already polled data.

The reason that the persistent storage of the data is not easy to accomplish is, that the relevant APIs which would usually be used for this (Datastore, Memcache, ...) use the context of the current Web request to provide additional information to underlying services. For example, Datastore uses the information of the User API to be able to save data on a per-user basis instead of just for the whole application. The User API in turn uses the information provided by the environment, like cookies of the request, etc. to gain information about if a user is logged in and which user that is. This would not be a problem at all if not for the need of threading required by history data polling. In order to regularly acquire history data from objects on the gateway, a threaded model is used in this API. The individual threads are initiated by user requests to the site and the following calls to the API. When the user initiates the recording of a history for an object, all the session information is present in this call, but the information is not available to the threads that get started and run separate from the request thread of the Web server. So if a worker-thread that records history data, polls an objects state and wants to save that state via the Datastore API, the Datastore API cannot acquire user information from the User API because no request context is available anymore in the local environment of the thread. To fix this problem, a few possible options are available to the programmer, three of which are discussed in Chapter 6.

Three basic methods have been implemented by which the history of an object can be recorded, each has its benefits in some situations. Those recordings can be initiated and stopped via the appropriate API calls available to the Web application developer. The recordings are currently stored using only the location of the objects as identifier. The location consists of host, port and *href* of the object. History data can be recorded by either:

1. Polling the object regularly and saving the current state of the object. This type of polling works by making regular GET requests for the object itself and then storing the returned object. The object gets stored whether it has changed its state since the last poll or not, so this type should only be used if regular polling and (possibly redundant) state recording is desired or the object changes states rapidly and therefore generates only a small amount of redundant data.

2. Regularly invoking `pollChanges` of a given Watch object to just gather recent changes. This mode takes the location of an oBIX Watch object and polls the changes of this Watch

object regurlarly to record changes made to the object between requests. The timeout between polls is based on the given Watch's lease time. The algorithm first tries to set the lease time as high as possible to reduce server load, the new lease time is calculated as follows: `Math.max(STANDARD_LEASE_TIME, WATCH_LEASE_TIME)`. Regardless of if this write to the Watch lease time works or fails, it then sets the timeout of the poll-loop to `0.8` times the actual lease time (old lease time if the write failed or the old lease time was higher than the standard, standard lease time otherwise) to ensure that the lease does not expire while the recording is active. Of course there are certain circumstances that can occur where the Watch can expire nonetheless (problems with connectivity, too high server load, etc.). In those cases, when the Watch does not exist anymore, the server will record an error object in the history and stop the polling of the Watch.

3. Creating a Watch object for a given history's feed object and invoking `pollChanges` regurlarly to gather changes made to the object via its history. This mode takes the location of an oBIX History object as an argument. It creates a new Watch object on the server by using the standard WatchService. The algorithm then adds the given History's Feed to the created Watch. The rest of the polling happens as in mode 2 (Watch polling).

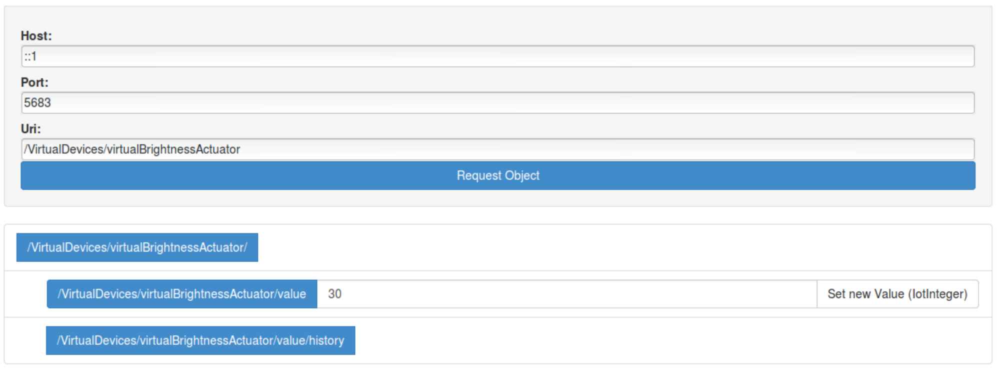
5.5 Evaluation

In the course of this thesis, the API has been tested with virtual devices and on lab hardware with the use of two simple demo applications which are able to modify objects' state and record history data for objects. As middleware application, the IotSyS Gateway was used. The API tests were conducted using the modified google testserver for local testing of Web applications. Through the demos, the testserver contacted the IoTSyS Gateway's oBIX server via CoAP to read, write and invoke objects.

With the demo application pictured in Figure 5.4, the user is able to read an object's state, including its child objects, traverse those child objects simply by clicking on them and modify the values of datapoints by simply changing the value and clicking the *Set new Value* button on the right. Listing 5.1 shows a small code fragment of the demo Servlet where the API is called and an object is retrieved. The call sequence through the layers of the API can be seen in Figure 5.5. For more detailed code samples, see Appendix A.

Listing 5.1: ModifyObjects API Calls

```
...
IotsysService is = IotsysServiceFactory
    .getIotsysService(getHost(), getPort());
...
//check if an object should get modified
if(getModifyUri() != null && getModifyUri().length() > 0) {
  IotObject modify = is.retrieveObject(getModifyUri());
  ... //set the value of the object based on it's type
  modify.write();
}
```


Host:

::1

Port:

5683

Uri:

/VirtualDevices/virtualBrightnessActuator

/VirtualDevices/virtualBrightnessActuator/

/VirtualDevices/virtualBrightnessActuator/value 30

/VirtualDevices/virtualBrightnessActuator/value/history

Figure 5.4: Demo for Modifying Objects

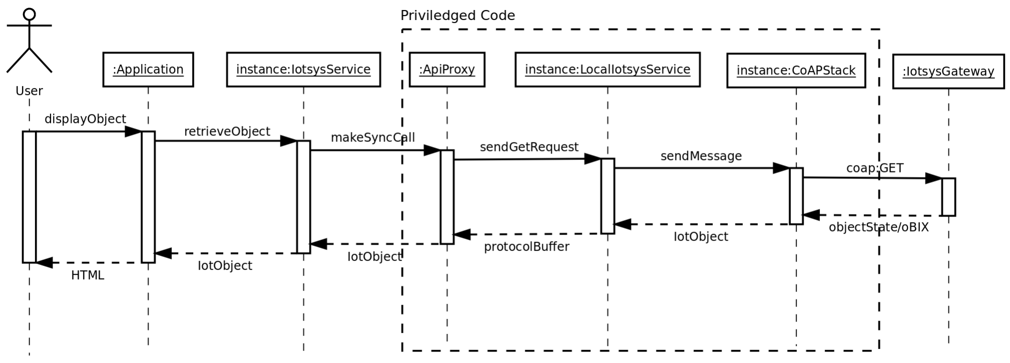
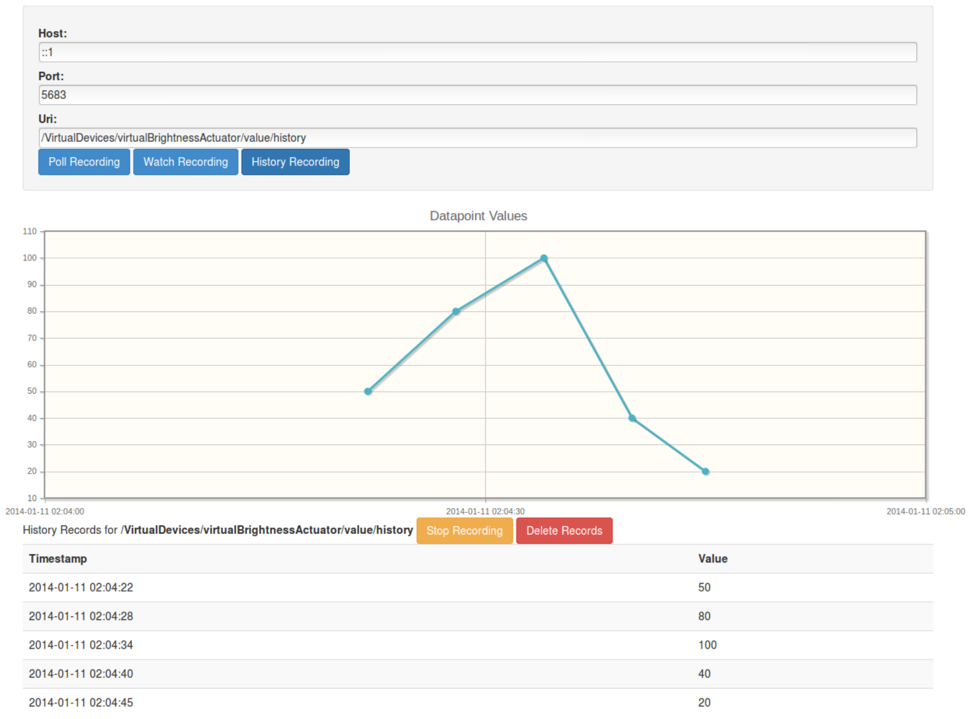


Figure 5.5: Call Sequence for an API Call

The demo shown in Figure 5.6 is able to record history data of specified objects in the three different ways that the API supports, regular polling of an object's state, adding an object to a watch and monitoring an object by watching its history, existing history data can be deleted and a recording for an object can be stopped by clicking the appropriate buttons. Both the code for the API and the code for the demos can be found in the IoTSyS project [1].

[1]`https://code.google.com/p/iotsys/`



Host:
::1

Port:
5683

Uri:
/VirtualDevices/virtualBrightnessActuator/value/history

History Records for **/VirtualDevices/virtualBrightnessActuator/value/history**

Timestamp	Value
2014-01-11 02:04:22	50
2014-01-11 02:04:28	80
2014-01-11 02:04:34	100
2014-01-11 02:04:40	40
2014-01-11 02:04:45	20

Figure 5.6: Demo for Recording History Data

CHAPTER 6

Conclusion and Outlook

Further Plans

The basic functionality of the API works well, but there is a still lot to be done in terms of additional functionality. Some of the more important features that would be desirable for such an API are listed below:

- Persistent storage of object history data. As of now, history data is only recorded in memory on the specific application server. This is due to the fact that session information is not preserved in a history thread's context. There are basically three ways to fix this issue. First, the context information that is used by other APIs to gain knowledge about the request can be identified and transferred to the thread. When the thread makes basic calls to APIs, it would have to insert that context information somehow in its environment. With this approach, there is no way that the context information can be passed to the APIs manually, so the method by which the APIs gain the information has to be identified and simulated in non-request API calls. This approach may require modification of the server's code. It uses the standard API methods that are exposed to a Web application developer via the respective `{ApiName}Service`. Another approach would be to identify the information necessary for the backend services' (`Local{ApiName}Service`) methods. This would mean going through the Appscale-specific code of the service-classes and identifying the methods used to communicate with the database service. The backend service classes (e.g. `LocalDatastoreService`) could then potentially be used directly from the thread without using the developer-friendly service classes (e.g. `DatastoreService`). In this approach, all the context information necessary could be given directly to the methods of the backend-service, but it would of course have to be acquired manually beforehand. The third feasable approach to use the underlying database system provided by Appscale is to communicate with the database server directly from the history-worker threads. This would mean acquiring all request context information manually, obtaining the right database server's address information (possibly via Zookeeper) and directly making requests to that server, probably via HTTP and appropriate Protocol Buffers.

- Support for the group-comm multicast service for oBIX objects, provided by the IotSyS project.
- Capability to keep ports open for the gateway to push information to. This would allow for a lot of new functionality to become available, like observing and alarming (see below).
- Support for actions that can be triggered if an object enters a specified state, either checked when polling the object regularly or via a push mechanism. Possible actions include, sending the user an e-mail or notifying him via SMS.
- Support for creating automation routines, the user would be able to create personalized routines for his home based on time and/or state of certain objects.
- Support for more protocols and formats.

Conclusion

In conclusion, the API provides basic functionality, but lacks more involved methods for processing IoT data which would have been out of the scope of this thesis. The provided features are already quite useful for basic home automation needs, but require manual input for controlling the system. The proof-of-concept shows that the idea of a complete home automation API would be absolutely feasible for commercial PaaS providers. It seems that today, the demand for home automation is not high enough to provide an incentive for the commercial providers to implement such a system. But in coming years, as home automation advances and more people will have the need for such software, we will hopefully see a rise in the availability of easily configurable cloud-based home automation software.

Bibliography

[1] ZigBee Alliance. *ZigBee IP Overview*. Jan. 2014. URL: http://www.zigbee.org/Specifications/ZigBeeIP/Overview.aspx.

[2] Amazon.com. *Amazon Elastic Compute Cloud*. Jan. 2014. URL: http://aws.amazon.com/ec2/.

[3] Amazon.com. *AWS Elastic Bean Stalk*. Jan. 2014. URL: http://aws.amazon.com/elasticbeanstalk/.

[4] Amazon.com. *AWS Elastic Compute Cloud*. Jan. 2014. URL: http://aws.amazon.com/ec2/.

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A Survey". In: *Comput. Netw.* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.05.010.

[6] *Carriots Homepage*. Jan. 2014. URL: https://www.carriots.com/.

[7] Open Geospatial Consortium. *OGC Sensor Web Enablement Page*. Jan. 2014. URL: http://www.opengeospatial.org/projects/groups/sensorwebdwg.

[8] Microsoft Corp. *Windows Azure Virtual Machines*. Jan. 2014. URL: http://www.windowsazure.com/en-us/services/virtual-machines/.

[9] Microsoft Corp. *Windows Azure Web Sites*. Jan. 2014. URL: http://www.windowsazure.com/en-us/services/web-sites/.

[10] C. Doukas and I. Maglogiannis. "Bringing IoT and Cloud Computing towards Pervasive Healthcare". In: *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. July 2012, pp. 922–926. DOI: 10.1109/IMIS.2012.26.

[11] Apache Software Foundation. *Apache CloudStack Homepage*. Jan. 2014. URL: http://cloudstack.apache.org/.

[12] Apache Software Foundation. *Zookeeper Homepage*. Jan. 2014. URL: http://zookeeper.apache.org/.

[13] OPC Foundation. *OPC Foundation Home Page*. Jan. 2014. URL: http://www.opcfoundation.org/ua/.

[14] M. Freund et al. "JSUA - An OPC UA JavaScript framework". In: *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*. Sept. 2013, pp. 1–4.

[15] W3C Semantic Sensor Network Incubator Group. *W3C SSN-XG Homepage*. Jan. 2014. URL: http://www.w3.org/2005/Incubator/ssn/.

[16] W. He, G. Yan, and L. Xu. “Developing Vehicular Data Cloud Services in the IoT Environment”. In: *Industrial Informatics, IEEE Transactions on*, pp. 1–1. DOI: 10.1109/TII.2014.2299233.

[17] Heroku. *Heroku Homepage*. Jan. 2014. URL: https://www.heroku.com/.

[18] IBM. *MQTT V3.1 Protocol Specification*. Jan. 2014. URL: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

[19] IETF. *CoAP Core Draft*. Jan. 2014. URL: https://datatracker.ietf.org/doc/draft-ietf-core-coap/.

[20] IETF. *IETF LoWPAN draft*. Jan. 2014. URL: http://tools.ietf.org/html/rfc4944.

[21] Eucalyptus Systems Inc. *Eucalyptus*. Jan. 2014. URL: http://www.eucalyptus.com/.

[22] Google Inc. *Gmail Homepage*. Jan. 2014. URL: https://mail.google.com/.

[23] Google Inc. *Google Appengine*. Jan. 2014. URL: https://appengine.google.com/.

[24] Google Inc. *Google Compute Engine*. Jan. 2014. URL: https://cloud.google.com/products/compute-engine/.

[25] Google Inc. *Google Docs Homepage*. Jan. 2014. URL: https://docs.google.com/.

[26] Google Inc. *Protocol Buffers on Google Developers*. Jan. 2014. URL: https://developers.google.com/protocol-buffers/.

[27] LogMeIn Inc. *Xively Homepage*. Jan. 2014. URL: https://xively.com/.

[28] C. Jennings et al. *SenML Draft*. Jan. 2014. URL: http://tools.ietf.org/html/draft-jennings-senml-08.

[29] Du Jiang and Chao ShiWei. “A study of information security for M2M of IOT”. In: *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*. Vol. 3. Aug. 2010, pp. V3–576–V3–579. DOI: 10.1109/ICACTE.2010.5579563.

[30] B. Khoo. “RFID as an Enabler of the Internet of Things: Issues of Security and Privacy”. In: *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*. Oct. 2011, pp. 709–712. DOI: 10.1109/iThings/CPSCom.2011.83.

[31] Matthias Kovatsch. *Californium Repository on Github*. Jan. 2014. URL: https://github.com/mkovatsc/Californium.

[32] G. Kulkarni et al. “A security aspects in cloud computing”. In: *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on*. Aug. 2012, pp. 547–550. DOI: 10.1109/ICSESS.2012.6269525.

[33] T. Le Guilly et al. "HomePort: Middleware for heterogeneous home automation networks". In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*. Mar. 2013, pp. 627–633. DOI: 10.1109/PerComW.2013.6529570.

[34] A. Lo, Yee Law, and M. Jacobsson. "A cellular-centric service architecture for machine-to-machine (M2M) communications". In: *Wireless Communications, IEEE* 20.5 (Oct. 2013), pp. 143–151. ISSN: 1536-1284. DOI: 10.1109/MWC.2013.6664485.

[35] A. Maiti and S. Sivanesan. "Advertising in location-aware cloud based home automation systems". In: *Remote Engineering and Virtual Instrumentation (REV), 2012 9th International Conference on*. July 2012, pp. 1–3. DOI: 10.1109/REV.2012.6293137.

[36] A. Maiti and S. Sivanesan. "Cloud controlled intrusion detection and burglary prevention stratagems in home automation systems". In: *Future Internet Communications (BCFIC), 2012 2nd Baltic Congress on*. Apr. 2012, pp. 182–186. DOI: 10.1109/BCFIC.2012.6218000.

[37] NIST. *Cloud Computing Definition*. Jan. 2014. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[38] *Observations and Measurements*. Jan. 2014. URL: http://www.opengeospatial.org/standards/om.

[39] OASIS Open. *oBIX Home Page*. Jan. 2014. URL: http://www.obix.org/.

[40] *OpenStack Homepage*. Jan. 2014. URL: https://www.openstack.org/.

[41] Pivotal. *Cloud Foundry Homepage*. Jan. 2014. URL: http://www.cloudfoundry.com/.

[42] Tom Preston-Werner. *GOD Homepage*. Jan. 2014. URL: http://godrb.com/.

[43] M. Rouached, S. Baccar, and M. Abid. "RESTful Sensor Web Enablement Services for Wireless Sensor Networks". In: *Services (SERVICES), 2012 IEEE Eighth World Congress on*. June 2012, pp. 65–72. DOI: 10.1109/SERVICES.2012.48.

[44] *SensorML*. Jan. 2014. URL: http://www.opengeospatial.org/standards/sensorml.

[45] Z. Shelby et al. *IPSO Application Profile Draft*. Jan. 2014. URL: http://www.ipso-alliance.org/wp-content/media/draft-ipso-app-framework-04.pdf.

[46] SicsthSense. *SicsthSense Homepage*. Jan. 2014. URL: http://sense.sics.se/.

[47] Appscale Systems. *Appscale Homepage*. Jan. 2014. URL: http://www.appscale.com/.

[48] Honbo Zhou. *The Internet of Things in the Cloud: A Middleware Perspective*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 2012. ISBN: 1439892997, 9781439892992.

APPENDIX A

Code Excerpts from the Demo Applications

Listing A.1: Base Servlet; used in both demos

```java
public abstract class BaseServlet extends HttpServlet {

  ... //parameter handling

  public void doGet(HttpServletRequest req,
      HttpServletResponse resp)
      throws IOException {
    ... //set parameters
    ... //set meta information
    writeBody(req, resp);
    ... //add js functionality
  }

  //various convenience methods

  /* used by the demo classes to write their body */
  public abstract void writeBody(
      HttpServletRequest req,
      HttpServletResponse resp)
      throws IOException;

}
```

Listing A.2: ModifyObjects demo Servlet

```java
public class ModifyObjectsServlet extends BaseServlet {

  @Override
  public void writeBody(HttpServletRequest req,
      HttpServletResponse resp)
      throws IOException {
    ... //check parameters
    IotsysService is = IotsysServiceFactory
        .getIotsysService(getHost(), getPort());
    try {
      //if an object should get modified
      if(getModifyUri() != null && getModifyUri().length() > 0) {
        IotObject modify = is.retrieveObject(getModifyUri());
        //set value of the retrieved object based on parameters
        if(modify instanceof IotInteger) {
          ((IotInteger) modify).setValue(Long.valueOf(getValue()));
        } else if(modify instanceof IotReal) {
          ...
        } ...
        modify.write();
      }

      //print the object that should be displayed to the user
      IotObject requested = is.retrieveObject(uri);
      if(requested instanceof IotError) {
        writeError(out, requested.getName());
        return;
      }

      writeObject(out, requested);
    } catch (CommunicationException e) {
      writeError(out, e.getMessage());
    }
  }

  private void writeObject(PrintWriter out, IotObject object) {
    ... //print html code to display object
  }

  ... //various convenience methods
}
```

Listing A.3: RecordHistory demo Servlet

```java
public class RecordHistoryServlet extends BaseServlet {

  @Override
  public void writeBody(HttpServletRequest req,
      HttpServletResponse resp)
      throws IOException {

    ... //check parameters

    IotsysService is = IotsysServiceFactory
        .getIotsysService(getHost(), getPort());
    try {
      IotObject obj = new IotObject();
      obj.setHost(getHost());
      obj.setPort(getPort());
      obj.setHref(getUri());

      if(getStop()) {
        is.stopHistoryRecording(obj);
      }
      if(getDelete()) {
        is.deleteHistoryRecords(obj);
      }
      if(!getStop() && !getDelete() &&
          !is.hasActiveHistoryRecording(obj)) {
        if("watch".equals(getType())) {
          is.recordWatch(obj);
        } else if("history".equals(getType())) {
          is.recordHistory(obj);
        } else {
          is.recordObject(obj, 30000);
        }
      }

      writeHistoryPlot(out);

      if(is.hasHistoryData(obj)) {
        List<IotHistoryRecord> records;
        records = is.getHistoryData(obj);
        writeHistoryData(out, obj, records);
      } else {
        writeHistoryData(out, obj,
            new ArrayList<IotHistoryRecord>());
      }
    } catch (CommunicationException e) {
      writeError(out, e.getMessage());
    } catch (NoSuchHistoryException e) {
      writeError(out, e.getMessage());
    } catch (HistoryExistsException e) {
      writeError(out, e.getMessage());
    }
  }
  ... //various convenience methods
}
```