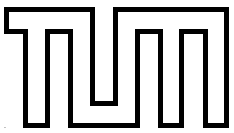


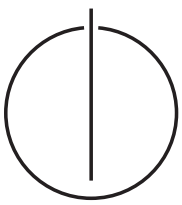
Fakultät für Informatik

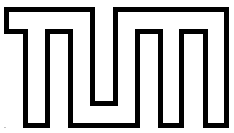
der Technischen Universität München

Bachelor's Thesis in Computer Science

Personalized Energy Measurements

Wolf-Steffen Rödiger





Fakultät für Informatik

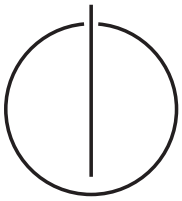
der Technischen Universität München

Bachelor's Thesis in Computer Science

Personalized Energy Measurements

Personalisierte Energiemessung

Submitted By:	Wolf-Steffen Rödiger
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisor:	Björn Schwerdtfeger (TU München) Markus Weiß (ETH Zürich)
Submission Date:	September 15, 2009



I assure the single handed composition of this bachelor's thesis only supported by declared resources.

Munich, September 15, 2009

. . . . . .
Signature

Ich versichere, dass ich diese Bachelorarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. September 2009

. . . . . .
Unterschrift

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Acknowledgements

This work has been conducted at the Bits to Energy Lab of the ETH Zurich and as such was supported by several industry partners. Special thanks go to my advisor Markus Weiß for his ongoing support but also to Tobias Graml, Claire-Michelle Loock, Thorsten Staake and all the others working at SEC 7 E for their help and a gorgeous time in a lovely city.

Abstract

In this thesis I propose the eMeter infrastructure which enables consumers to measure their personal energy consumption and thus helps them to conserve energy. It is designed to be as lightweight and easy-to-use as possible. For this purpose it uses one of the upcoming smart electric meters as a single sensor and presents the user interface on a mobile phone. Users can keep track of their entire house consumption as well as measure the consumption and cost of single appliances.

Zusammenfassung

In dieser Arbeit stelle ich die eMeter Infrastruktur vor, die es Verbrauchern ermöglicht ihren persönlichen Energieverbrauch zu messen und ihnen somit hilft Energie zu sparen. Das System wurde so gestaltet, dass es möglichst leichtgewichtig und einfach zu bedienen ist. Aus diesem Grund wird einer der zukünftig standardmäßig verbauten, fernauslesbaren Stromzähler zur Messung des Stromverbrauchs verwendet und die Benutzerschnittstelle auf einem Mobiltelefon präsentiert. Benutzer können sich über den Verbrauch ihres gesamten Haushalts informieren als auch die Kosten und den Verbrauch einzelner Geräte messen.

Contents

1 Introduction

At 3 o'clock in the afternoon on September 4, 1882, Thomas Edison opened the first central power plant in the United States. It was named Pearl Street station and could serve one square mile of New York City with its produced 600 kilowatts, enough to power 7200 lights.

A major challenge besides the generation and distribution of electricity was to design a way to bill customers on their consumption. Instruments to measure the flow of current had been available since the early nineteenth century. However, working electric meters which could also record power consumption were not available until spring 1882.

Edison did not charge his customers until the whole system was operating successfully, which took some additional time. Finally, the first electric bill was sent to the Ansonia brass and copper company on 18 January 1883. It amounted to $50.44 [21].

1.1 Lack of transparency in electricity consumption

Since the times of Thomas Edison electricity consumption has been mostly obscure to the consumer. It is not possible to clearly identify cause-and-effect relationships based on an electric bill delivered only once a year–long after the time of consumption.

Additionally, the bill only states the total amount so there is no way to determine times of high usage or even single appliance consumption. Residential energy use is different from other consumer goods in the way that it remains abstract, invisible and intangible to the consumer [12]. Electricity is seen as a necessity and not as a product because people simply use appliances and not energy [5].

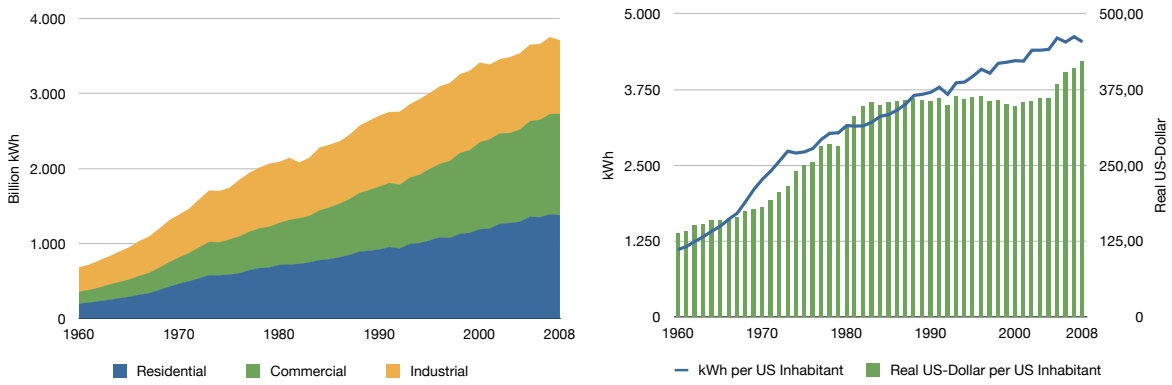
In 1977 Seligman and Darley illustrated this situation as follows: "Appliances are run, the air conditioning cycles, hot water is used, the furnace runs, and the homeowner has no way of determining what amounts of energy are used by these devices. The utility bill that the homeowner gets is not analyzable into these components. Nor does the bill appear close in time to the energy usage" [37].

Even after more than a hundred years of electric power supply, the consumer knows little about his power consumption over time and even less on the outcome of specific device usage schemes. However, this knowledge is badly needed to empower users to conserve energy in the times of global warming and increasing threats to biodiversity [14].

Households constitute an important target group for energy conservation, being major electricity end-users and, consequently, contributors to global warming [1]. In the year 2008 the residential sector was responsible for 37 % of the electricity end use in the United States [44]. Figure 1.1 depicts the increase of residential power consumption and cost over the last 50 years.

1.2 Recent developments

Several recent technological and political developments are crucial for the approach presented in this work. Most important is the progress in computing hardware, often characterized as Moore's Law. The increasing processing power and at the same time decreasing size of computers has lead to the development of powerful mobile computing platforms. Smart phones for instance are able to present interactive and appealing user interfaces.



(a) Electricity end use by sector

(b) Per capita electric consumption and cost

Figure 1.1: History of energy consumption in the US [34, 35, 44].

Another important development has taken place in the field of electric meters which has lead to so-called smart meters which are able to identify consumption in more detail and support telemetering[1]. Smart meters have already been adopted to a great extent in Italy, Canada, the United States and Turkey. The largest deployment was undertaken by the dominant utility in Italy which provided their entire customer base, over 27 million customers, with smart meters [53].

The distribution of smart meters is mainly driven by regulation. The EU for instance states that "[b]illing on the basis of actual consumption shall be performed frequently enough to enable customers to regulate their own energy consumption" [10]. Conforming with this directive, smart meters are required by regulation in new and renovated buildings in Germany as of 2010 [16].

At the moment it seems that smart meters will replace classical electric meters in the next few years. This will lead to more informative bills including comparisons with previous consumption periods and the average user in the same user category [10]. Furthermore, it will enable the development of computerized, real-time and interactive feedback which can help to make the consumer more aware of his energy consumption. However, privacy issues concerning the consumption data are sill relevant [17].

1.3 Intention of this work

This work has been conducted within an industry project cooperation at the Institute of Pervasive Computing, ETH Zurich[2]. Its goal is to present and discuss the eMeter infrastructure which allows users to get feedback on their electricity consumption in real-time. The system has been set up in a laboratory environment at the Bits to Energy Lab[3]. It builds upon a standard smart meter which measures the current load every second.

This work consists of the following parts: At first I will present related work to classify the proposed solution. The next chapter discusses best practices for the design of feedback on electricity consumption. These guidelines served as the theoretical foundation for decisions made while designing the eMeter system. Main focus lies on section four which describes the architecture in detail. At last I will discuss the results and give an outlook on future work.

[1] Telemetering denotes the possibility to communicate the measured electric consumption via some network back to the local utility for monitoring and billing purposes [53].

[2] `http://www.vs.inf.ethz.ch`

[3] `http://www.bitstoenergy.ch`

2 Related work

I will first give a short summary of the research on energy feedback which has its seeds in the energy crisis of the 1970s. This is followed by a discussion of different approaches to energy consumption displays and a short summary of commercial products. At last I will point out the originality of the eMeter system.

2.1 History of research on energy feedback

The beginning of research on energy feedback dates back to the 1970s. Triggered by the 1973 "oil price shock"[1], increased political effort and scientific research had gone into energy conservation. The energy crisis raised the concern about a possible exhaustion of fossil fuels, which was the dominant reason for research then [14]. In the United States community-based workshops on energy conservation were held regularly [15] and TV and radio spots with the same issue were broadcasted [25].

In 1977 Hayes and Cone [18] identified two major ways to deal with the problems caused by high levels of consumption in the industrialized nations of the world. Namely these were the development of new sources of energy and the change of the energy consumption behavior. Following the second approach they studied the influence of monetary payments, energy information and daily feedback on the electricity use in four units of a university student housing complex. It turned out that paying for reductions was the most effective approach. Of the two other procedures feedback did appear to them to be more promising than information because the latter resulted only in a temporary effect. However, at that time "information in the form of massive educational campaigns seem[ed] [to be] the main strategy adopted by governmental agencies and power companies to control the consumption of energy" [18].

In 1981 Geller came to a similar conclusion: "Workshop attendees showed very few applications of the energy-conservation strategies emphasized at the workshop" [15]. He also stressed the importance of feedback: "Significant reductions [...] resulted when informational programs were supplemented [...] with frequent feedback" [15].

Since then many studies investigated the effect of different kinds of feedback on energy consumption. In 2008 Fischer [12] published a review that covers five review studies and 21 original papers about this topic. She comes to the conclusion that "[o]ne result, at least, seems clear: feedback stimulates energy and specifically, electricity savings. [...] Usual savings are between 5 and 12 %" [12]. Darby presented a similar result two years earlier: "The norm is for savings from direct feedback [...] to range from 5–15 %" [6].

Hayes and Cone proposed already in 1977 that "[f]eedback could b[e] maximized by an online, in-house meter [...]. It would seem feasible to build a meter to report continuously the amount of energy [...] consumed for the day and month to date, project a total monthly bill [...], and turn on an indicator when certain consumption levels were exceeded" [18]. This can be seen as an early description of energy consumption displays which I will discuss next.

[1] The 1973 oil crisis started in October 1973, when the members of the OAPEC proclaimed an oil embargo "in response to the U.S. decision to resupply the Israeli military" during the Yom Kippur war; it lasted until March 1974 [43].

2.2 Feedback devices

In theory all homes already have a continuous energy consumption indicator: the classic electric meter. However, this is generally not designed for consumers to monitor their power usage. As Wood puts it in 2003: "Conventional meters have relatively crude displays, which tend to dissuade householders from using them as energy-saving tools" [55]. This is supported by a study carried out by Meyel in 1987 [29] which points out that people often do not understand their energy meters. More than 50 % did not know where their meters were and 45 % could not read them.

Special devices for feedback on electricity consumption have been designed and studied since the 1970s. They can be divided into those which inform users about the power consumption of the entire house and others which provide details on single devices.

2.2.1 Entire house feedback

In 1976 Kohlenberg et al. [25] designed a very simple feedback device on electricity use. It consisted of a current-sensitive relay installed on the power line to the home and a 40 W light bulb placed in the kitchen. Every time the current level exceeded 90 % of the peak levels recorded over the last two weeks, the light would turn on. This very simple feedback device already provided some insights into power consumption: "All three families indicated surprise at the relationship between activation of the signal light and operation of the hot-water heater" [25]. As a first feedback device the signal lamp as such was stationary and did not provide information besides the fact that a peak load had occurred.

In 1979 McClelland and Cook [28] studied the effect of feedback devices which displayed electricity consumption continuously in cents per hour. This lead to "an average electricity savings of 12 %" [28] and was one of the first evidences that feedback devices enable major savings in power usage.

Sexton et al. [38] investigated the consumers response on Continuous-Display Electricity-Use Monitors (CDEUMs) in 1987. They evaluated a monitor which could display the current rate of electricity use in cents per hour, the projected monthly electricity bill and the amount of the last bill. It also had an indicator light, blinking if a user defined monthly budget was likely to be exceeded. They came to the result that "comprehensive information like that provided by CDEUMs [could] significantly affect consumer behavior and speed adjustment to significant shocks in economic signals" [38].

In 2004 Petersen et al. [33] developed an automated data monitoring system that provided dormitory residents with real-time web-based feedback on power use. The current consumption was measured with off-the-self electricity sensors, sent every 20 seconds wirelessly to a server and saved in a database. Residents could track their consumption by calling a web site on their computer. This site included gauges which displayed the current load and graphs for the last day and week. With its 20 second update time this solution comes near to real-time feedback which is realized in the eMeter system by measuring the consumption every second.

Yun [56] investigated the impact of minimalist in-home energy consumption displays (ECDs) in 2009. A sensor clip for the fuse box is used to measure the entire home consumption and communicates this data wirelessly to a portable and a stationary display. The displays use simple bar graphs which consist of four LED lights and visualize the current load from 0 to 4096 W in steps of 4 W since one LED has 256 levels of brightness. They came to the result that the portable ECDs were used like stationary ones after a phase of testing different devices all over the home. They concluded that "while our device was portable, it did not provide true mobility in the sense of being practical to wear or carry at all time" [56]. The proposed eMeter sys-

tem tries to provide this kind of "true mobility" by implementing the user interface on a mobile phone. Most mobile phone users seem to find it practical to wear their phone all the time. As a consequence, wherever the user may be when he wants to measure a device or look at the current consumption, he most probably has the required device at hand.

In the same year Petersen et al. [32] presented a web-based prototype of a mobile application for the iPhone called Wattbot. It displays the totaled consumption of each circuit of the home for a chosen time span. The data is collected through sensor clips, one for each circuit and then broadcasted wirelessly to the phone. This approach is opposite to the one chosen for the eMeter system. Instead of installing more sensors and requiring manipulations in the electric box, the eMeter system builds on a standard smart meter as a single sensor.

2.2.2 Device level feedback

The first more sophisticated design of an electrical energy monitor which provided details on individual devices was described by Hunt et al. in 1986 [20]. Their goal was to built "a device that can conveniently measure, display and control the energy used by various appliances in the home" [20]. The system consists of a master unit which displays the power consumption and communicates to the slave units over power line [2]. The data is acquired every 10 minutes. Slave units are plugged inline between devices and outlets to remotely control them and measure their consumption. The master unit displays the total daily consumption, month to date consumption and a bar graph of the consumption of the last 13 months in total or for each device separately.

In 2002 Ueno et al. [42] designed an energy-consumption information system. It consists of one sensor which measures the consumption of the entire house, several other sensors that measure single appliances and an informational terminal. The terminal displays various details including energy saving tips, load-curves and historic comparisons. A conducted study revealed that the power consumption of the monitored appliances was reduced by about 12 %.

In 2007 Lifton et al. [27] introduced a sensor network which consists of a set of power strips able to measure sound, light, electrical current, voltage, vibration, motion and temperature. They visualized the data from a deployment of 31 nodes on a simple map as well as in 3D in Second Life. Metaphors were used to represent data which usually remains invisible. Fires of different sizes for instance disclosed the current energy consumption.

Hinterbichler presented 2008 [19] an HCI approach to the design of energy consumption indicators (ECIs) because he "believes the interfaces of most ECIs do not adhere to HCI principles and are deficient in their current forms" [19]. The newly constructed system uses several Watts Up [8] devices as sensors which are connected to a laptop via USB which also served as the display. The visualization provides two cumulative meters one for the current week and the other for the current load. Additionally it provides a breakdown into single appliances as well as energy saving tips. A history view provides bar graphs of the last six weeks or months and the last hour. A community view provides comparisons with the consumption of neighbors. The data is updated every minute.

2009 Jiang et al. [22] constructed a wireless sensor network for real-time AC metering called ACme. Several ACme nodes are plugged between outlets and devices. They act as sensors for the power consumption and send their readings every minute to the ACme application. This in turn consists of a web front-end, a database back-end, and a daemon process. The daemon fills the database which is used by the web front-end to present the data in tabular and graphical form.

[2] Power line carrier communication is a system for carrying data on a conductor also used for electric power transmission [51].

A different approach for device level power monitoring was implemented by Kim et al. in 2009 [24]. Their ViridiScope system does not require inline sensors which have to be installed between the standard AC plug of a device and the outlet. The motivation to avoid this is that "some of the major energy consumers [could] not be easily instrumented [this way]. For example, most heating and ventilation systems (HVAC) and electric boilers do not have standard AC plugs, or are hard-wired to the main power lines" [24]. The system instead uses indirect sensing of measurable signals which are emitted by appliances when consuming energy. A PC is responsible for the fusion of data from the utility meter and the distributed sensors which are magnetometers, light sensors and microphones. "Experiments show that the system tracks the per-appliance power consumption to less than 10 % error" [24].

2.2.3 Commercial products

Several products which present information on the entire house or device level consumption are commercially available.

Entire house sensors can be differentiated by the means of data acquisition. Most available systems use sensor clips which have to be installed in the electric box of the house. This is not possible in many European countries. The Blue Line PowerCost Monitor [4] is an exception to this as it uses image recognition technology to read the classic electric meter automatically. Another distinguishing feature between products are the means of communication. The Blue Line Powercost Monitor uses wireless communication which can be affected by walls and ceilings especially if the electric meter is located in the basement. The data is refreshed only every 30 seconds. The Energy Detective [9] updates every second and uses power line communication. While this avoids the problems of wireless communication it has to deal with noise and interferences of other power consumers since the electric circuits were not originally intended for communication purposes. Wattson [7] uses a sensor clip and wireless communication but has an additional unique feature: It glows according to the current consumption. This can be more intuitive than bare numbers on an LCD display which is the way other devices display their data.

All device level sensors currently available come as inline sensors which must be plugged between device and outlet. Kill A Watt [30] and Watts Up [8] display the consumption and cost of only one connected device with their LCD display. The UFO Powerstrip [45] provides the capability to sense and remotely control more than one device at once by incorporating the form factor of a power strip. It is announced for late 2009 and will be operated by an iPhone application.

2.3 Yet another energy consumption display?

After this detailed description of already existing systems the question arrises if the eMeter system is just another energy consumption display. However, it has some distinct characteristics:

The eMeter system needs no other sensors besides a standard smart meter. This includes that no fuse box manipulation or extra cabling is required. The use of a mobile phone for the user interface has also many benefits: Users do not need to buy single purpose hardware which is either bound to a specific place by its power cable or has batteries which need to be replaced every so often. Mobile phones are at hand when needed and make it possible to explore the consumption of different appliances all over the home. Devices can be measured even if they have no standard AC plug or cannot be unplugged. Consumers can monitor the consumption of their home wherever they are when provided with access to the Internet. Best of all: the phone can be called in the case it got lost.

3 Design decisions

A study conducted in the 1990s evaluated the electric consumption of ten identical all-electric homes inhabited by at least two people and equipped with the same appliances. Despite all these commonalities, the energy usage varied by a factor of 2.6 [31]. This significant difference is a strong indicator that the residential behavior affects energy consumption largely. Indicating that "interventions to modify occupant behavior could result in appreciable energy savings" [36].

The design of feedback on energy consumption has influence on the change of consumer behavior. For this reason I reviewed different design commendations from Hinterbichler 2008 [19], Fischer 2008 [12] and Fitzpatrick and Smith 2009 [13] for the design of the eMeter system.

3.1 Data specific design issues

The main question when designing a feedback device is how to select and present electricity consumption data. This should be dealt with great attention as the right choices can raise consumer awareness and possibly lead to behavior change and thus energy conservation.

3.1.1 Frequency and duration

The more frequent feedback is given the more effective it is in terms of energy savings [12]. Real-time feedback makes the link between actions and effects clear and improves the understanding of the underlying correlations. "Immediate feedback could be very helpful while weekly to monthly feedback may be helpful, but is not sufficient [...] on its own" [12]. The duration of feedback is also important, because the formation of new habits takes time. Long term feedback therefore leads to more persistent savings [12].

The eMeter system measures every second and thus tries to achieve real-time feedback.

3.1.2 Breakdown

Most consumers are interested in the usage of individual appliances and not only the total consumption of the home [12]. However, measurement devices which have to be installed between device and outlet are often out of sight during normal use and proved to be the least popular device in the study conducted by Fitzpatrick and Smith [13]. In the end "participants were unanimous that a central location for immediate feedback was preferred to appliance-specific feedback devices" [13]. However, "participants found other engaging ways of understanding the impact of specific appliances—for example, by watching how the load changed when a kettle was turned on" [13].

From this it seems obvious to conclude that a mobile display should provide the possibility to measure single appliances while not being attached to them. For this reason the eMeter system enables users to measure devices by simply turning them on or off. After a few seconds the mobile phone interface specifies the consumption of the selected appliance. This has some drawbacks as it cannot accurately measure devices with varying consumption or devices which cannot be easily turned on or off. Yet Fitzpatrick and Smith state: "Usage by appliance might be desired in principle but not at a cost of a plethora of devices and not necessarily by increasing technical complexity. Design efforts might be better directed toward encouraging playful engagement [...]" [13].

3.1.3 Comparisons

With historical feedback consumers can compare their current use with historical data. On the contrary, comparative feedback provides the users with information on how they perform relative to others. Normative feedback in turn lets the consumers match custom goals. Comparisons can reveal "whether consumption in a certain period or of a certain household is 'out of the norm'" [12].

Volunteers of the study carried out by Fitzpatrick and Smith preferred historical over comparative and normative feedback. The authors conclude that "in a historical view, most wanted a bar graph representation with a click-through hierarchy of granularity" [13]. The proposed eMeter iPhone application provides bar graphs over the last five hours, days, weeks, months, quarters, half-years and years as well as profile graphs and projections of the current consumption.

Comparisons with other users can provide context and thus help categorize one's own usage better. However, this can also lead to undesired effects. Studies have found cases where efficient users actually increased their energy consumption significantly after realizing their below-average usage [12]. Personalized goals can circumvent this problem by providing consumers with goals fitting their own consumption. This also avoids the need to built adequate data bases for comparisons with other households [12].

Normative comparisons can help the user to understand what the displayed numbers mean by providing additional context. As Hinterbichler puts it: "Even with a visual display, if the information shown has no relevant context, it will be hard to interpret" [19]. The eMeter system uses the colors green, yellow and red when displaying the current load and historic values. This rating is based on the user-defined household size and the number of tenants. It is consistent with study participants "suggest[ing] traffic light colors as an additional information" [13].

3.1.4 Presentation

Feedback on power use must be easily understandable and interactive to effectively involve consumers. Displays with bare numbers can loose attention after an initial time of usage because they are too complex to be conceived with a quick look. Ambient displays which for example glow according to the consumption like Wattson [7] can inform the user of changes in consumption in an unobtrusive way. On the other hand users do not "appreciate abstract *ambient-only* displays, finding them ambiguous and lacking easily interpreted detail" [13]. Alarms which signal high consumption by playing sounds are as well undesirable because they are thought to be annoying [13].

The eMeter application presents the current power load with a visualization similar to the speedometer of a car because "graphical displays can attract the viewer and promote curiosity" [19]. The familiar metaphor enables the user to perceive the range of consumption even with a quick look. At the same time the actual wattage is also displayed as a number for more precise information. The measurement functionality encourages users to interact with their appliances and learn about their consumption. The proposed system uses different kinds of presentations. A graphical representation is used for the current load, bar graphs and line charts for the historic comparisons and tables for device details.

Although a mobile phone does have many advantages if used as a feedback device, it is not well suited to provide ambient feedback. To compensate this, the speedometer-like interface has colored ranges so the user can check with a glance if the consumption is in standby, low, middle or high usage range.

3.1.5 Metrics

There are several possible metrics in which energy consumption can be displayed. These

include power (kW), cost (currency) and CO_2 emission equivalents (tones). Fitzpatrick and Smith found out that "participants preferred cost over energy with CO_2 emissions the least preferred" [13]. However, this order depends on how monetary representations are used. The extrapolation of the current use into a money-per-year metric as realized by Wattson [7] was thought by some participants to be meaningless. In this case actual wattage was favored [13]. The significance of CO_2 emission in tones depends on the knowledge: "None of the participants [...] [did] know how to interpret CO_2 emission tones" [13].

The eMeter system uses kilowatts to express the current load and provides the possibility to choose between kilowatt-hours and cost in historic comparisons. The consumption of measured devices is expressed in cost and kilowatt-hours per year. The extrapolation is based on a user-defined usage plan. CO_2 equivalents are omitted for simplicity.

3.1.6 Additional information

An energy consumption display could display more feedback for the interested user. Hinterbichler [19] recommends to provide users with personalized energy saving tips which are based on the specific household. This could include commendations to replace inefficient devices. He states that "a home energy interface should not shy away from judging users when necessary, as this can potentially increase the interface's effectiveness" [19].

3.2 Device specific design issues

Even though it is important to decide how to select and visualize consumption data, issues concerning the design of the device and its hardware should not be neglected.

3.2.1 Form

When designing energy displays one has to keep in mind that such a device constitutes some kind of intrusion into the home. Hence "aesthetics and fitting into the home space become important" [13]. If insufficient attention is payed to this, devices can easily get abandoned by users. This happened to a plug-based device evaluated in [13] which was described by the participants to be "[...] ugly, unrewarding, and difficult to use" [13]. Fitzpatrick and Smith conclude that it "ultimately deemed unsuitable for the task with only one participant using it" [13].

Besides the pure aesthetics consumers also lay value on portability and energy efficiency so they can survey their home whenever they want without spending too much energy on the consumption display itself.

The eMeter system uses a mobile phone for the user interface which minimizes the intrusion into the home and provides a high degree of mobility and energy efficiency.

3.2.2 Location

The place of an energy monitor should be easily viewable to enable quick status checks. At the same time it should not use too much space. Participants of [13] wished they could mount the display to a wall to save surface space. Others wished they had the possibility to get feedback even outside the home provided through a mobile phone or web site.

The design decision to use a mobile phone in the eMeter system allows for status checks not restricted to the home and saves space but in turn does not provide the permanent viewable feedback of a fixed device.

3.3 Summary

Fischer states that "most successful feedback combines the following features: it is given frequently and over a long time, provides an appliance-specific breakdown, is presented in a clear and appealing way, and uses computerized and interactive tools" [12].

4 Architecture

In this chapter I will describe the architecture of the eMeter system and the prototype deployment in detail.

4.1 Overview

The eMeter system consists of three main components. Namely these are the parser, the web server and the iPhone application. The UML[1] component diagram depicted in Figure 4.1 shows the connections and dependencies among these three parts of the system. It visualizes the high-level structure as well as the provided and required interfaces of the single components.

4.1.1 The three components in brief

The parser is responsible for the communication with the smart meter. It reads the relevant measurement values out of messages which it requests regularly from the smart meter. The server preprocesses these measurements and stores them in a database. It provides access to the current load and additional aggregated data like the totaled consumption over different time spans. Finally, the eMeter iPhone application visualizes the information for the user. It provides also further functionality built on top of the consumption data.

4.1.2 Communication overview

The three components can communicate over public networks like the Internet, local area networks or even on the same machine without changes since they use simple TCP/IP connections. It is for example conceivable that the parser and server run on the same machine only accessible locally in the home. Another possibility would be that the server is also reachable from the Internet or even installed in a data centre. This would enable the user to monitor the consumption of his home whenever he has an Internet connection, independently from his location.

4.1.3 Architectural style

The eMeter system is built as a Client-Server-Architecture. The two clients, the parser and the iPhone application, are separated from the server by a uniform interface. The clients are not concerned how the data storage is realized which remains internal to the server. They only manipulate resources identified by Uniform Resource Identifiers (URIs). This improves the portability of the client code. The server knows little about the clients and their implementation or current application state. This makes the server simpler and more scalable. The decoupling of data acquisition, storage and visualization leads to a simple and extensible system. A new visualization like one for a TV or another data source providing for instance information on water consumption could be added without affecting the other components.

To be more precise about the architectural style, it can be described as being RESTful[2]. REST is explained in detail in Section 4.3.5.

[1] Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering [54].

[2] Representational state transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. Conforming to the REST constraints is often referred to as being 'RESTful' [52].

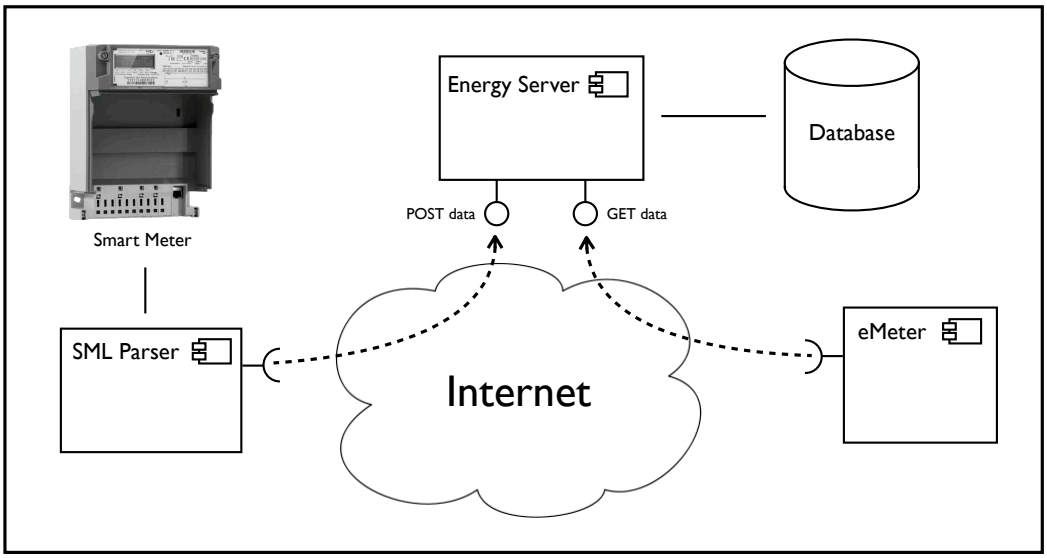


Figure 4.1: UML component diagram of the eMeter system.

4.2 SML parser

The SML parser and the server perform their work behind the scenes and thereby constitute the back-end. They are invisible to the user but fundamental for the functionality provided by the user interface. Furthermore they are responsible for the data acquisition from the smart meter, basic preprocessing and permanent storage. The SML parser is discussed in this section and the server in the next one.

A custom parser is needed because smart meters communicate with messages encoded in the especially designed smart message language (SML). The parser had to be built from scratch following the public SML specification [40]. It was developed in Java using the Eclipse IDE.

I will first discuss the different functional and non-functional requirements. Then I will describe SML, followed by a discussion of the parser's run-loop. After this I will present implementation details and state reasons for the decision in favor of a pull data acquisition strategy instead of push. At last I will discuss solutions to problems which were revealed during development.

4.2.1 Functional requirements

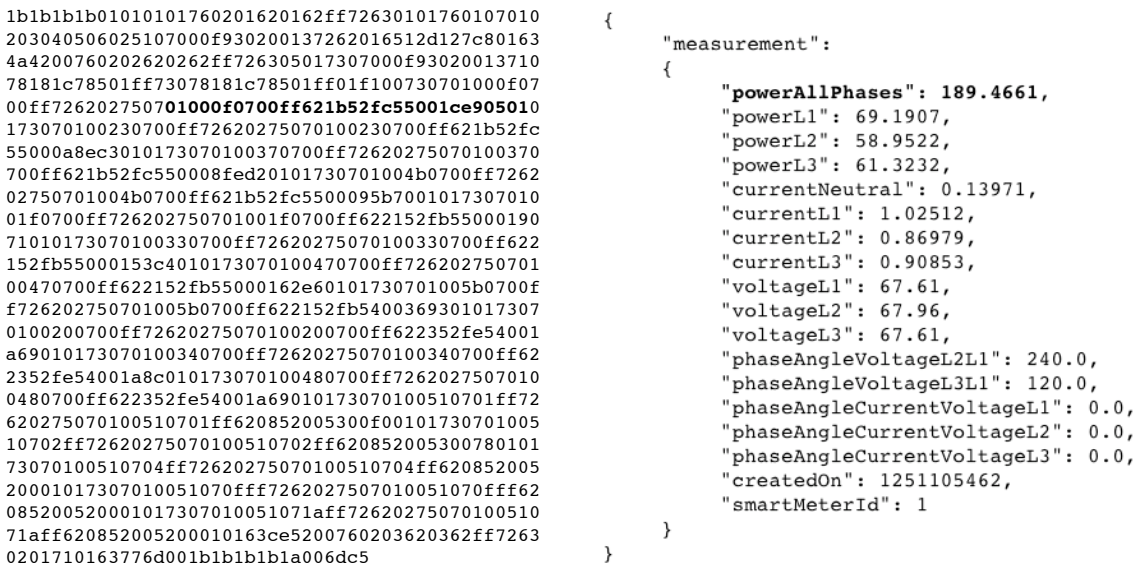
The parser has to provide three basic functionalities: First the measurement data has to be requested from the smart meter. Second the SML response has to be parsed and translated into an encoding the server understands. Finally the newly encoded data has to be sent to the server.

4.2.2 Non-functional requirements

Besides the functional requirements there are some more demanding requirements concerning the quality of operation.

Performance: Since the smart meter measures the current consumption every second the parser has to request, process and forward the data in less than a second. This is important to achieve real-time feedback which was one of the design decision discussed in Section 3.1.1. However, this requirement led to some unexpected problems as described in the section about problems and solutions.

Robustness: Communication over the Internet can be unreliable. The server could at some time be disconnected for maintenance reasons or the parser could loose its Internet connection eventually. For this reason the



```
1b1b1b1b0101010176020162016 2ff72630101760107010
203040506025107000f93020013726201651 2d127c80163
4a4200760202620262ff726305017307000f93020013710
78181c78501ff73078181c78501ff01f100730701000f07
00ff726202750701000f0700ff621b52fc55001ce90501 0
173070100230700ff726202750701002307 00ff621b52fc
55000a8ec3010173070100370700ff72620275070100370
700ff621b52fc550008fed20101730701004b0700ff7262
02750701004b0700ff621b52fc5500095b7001017307010
01f0700ff726202750701001f0700ff622152fb55000190
710101730701003307 00ff726202750701003307 00ff622
152fb55000153c4010173070100470700ff726202750701
00470700ff622152fb55000162e60101730701005b0700f
f726202750701005b0700ff622152fb5400369301017307
0100200700ff726202750701002007 00ff622352fe54001
a690101730701003407 00ff726202750701003407 00ff62
2352fe54001a8c0101730701004807 00ff7262027507010
0480700ff622352fe54001a690101730701005107 01ff72
6202750701005107 01ff620852005300f00101730701005
10702ff726202750701005107 02ff620852005300780101
730701005107 04ff726202750701005107 04ff620852005
200010173070100510 70fff726202750701005107 0fff62
085200520001017307010051071aff72620275070100510
71aff620852005200010163ce5200760203620362ff7263
0201710163776d001b1b1b1b1a006dc5
```

(a) SML encoded measurement (580 bytes)

```
{
    "measurement":
    {
        "powerAllPhases": 189.4661,
        "powerL1": 69.1907,
        "powerL2": 58.9522,
        "powerL3": 61.3232,
        "currentNeutral": 0.13971,
        "currentL1": 1.02512,
        "currentL2": 0.86979,
        "currentL3": 0.90853,
        "voltageL1": 67.61,
        "voltageL2": 67.96,
        "voltageL3": 67.61,
        "phaseAngleVoltageL2L1": 240.0,
        "phaseAngleVoltageL3L1": 120.0,
        "phaseAngleCurrentVoltageL1": 0.0,
        "phaseAngleCurrentVoltageL2": 0.0,
        "phaseAngleCurrentVoltageL3": 0.0,
        "createdOn": 1251105462,
        "smartMeterId": 1
    }
}
```

(b) Respective JSON representation (435 bytes)

Figure 4.2: Comparison of the original SML and the resulting JSON encoding.

parser has to be robust and able to recover from network errors.

Portability: At some time in the future the parser could be run on a low-cost embedded system or even on the smart meter itself. This requires that it is lightweight as well as executable under different operating systems and in different execution environments.

Maintainability: Adaption to new smart meters, communication ways or data formats should have little impact on unrelated code. This requires that the parser is built in a modular way.

4.2.3 Smart message language

The smart message language was designed by the SyM^2 consortium for the special purpose of communication with electric meters. The packed message format described by the SML specification is optimized "for use in both classical communication routes (PSTN, GSM, etc.) and in package-oriented network operation" [41].

SML makes electric meters smart in the way that it provides them with the possibility to exchange data with a remote meter reading system. In our case this remote meter reading system is represented by the parser and the corresponding computer which connects to the smart meter via a LAN port.

The parser was built considering version 1.02 of the specification which was published on January 19, 2008 [40]. It reads the binary coded messages outputted by the smart meter which have a size of about 580 bytes and translates them into a tree structure of Java objects. Finally a JSON[3] representation of selected attributes is sent to the server which has a size of about 450 bytes. A sample of a binary coded SML message describing a measurement and the respective JSON representation are depicted in Figure 4.2. The power over all phases is in both cases displayed in bold for the purpose of comparison.

[3] JSON, short for JavaScript Object Notation, is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects) [47].

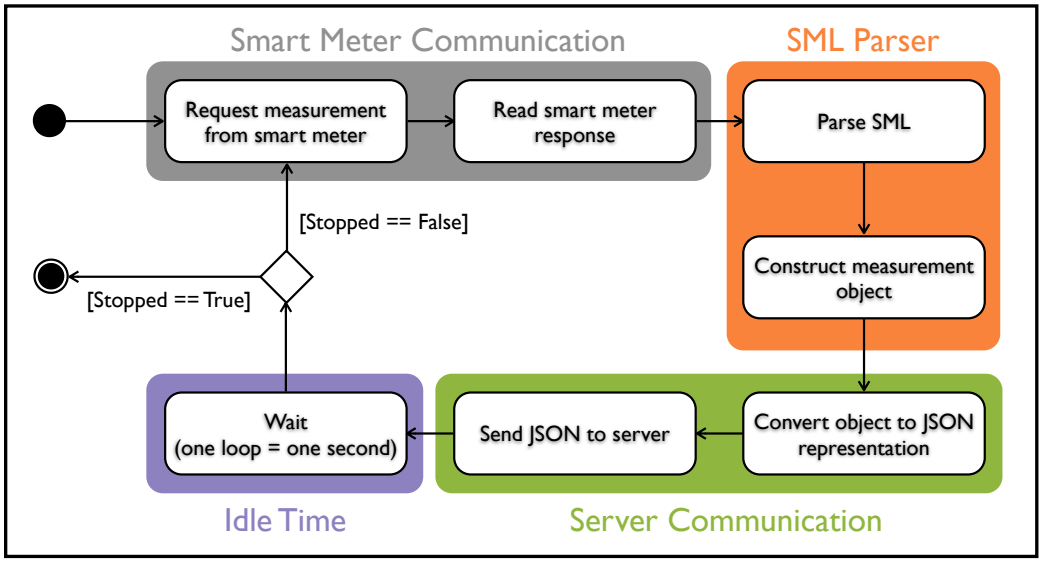


Figure 4.3: UML activity diagram of the parser's run-loop.

4.2.4 Run-loop

Figure 4.3 depicts the parser's run-loop. It can be divided into four phases: First the smart meter communication, second the SML parsing, third the server communication and fourth the remaining idle time.

In the first phase the parser connects to the smart meter via a TCP/IP socket. Once the connection has been established it sends a special SML file consisting of three SML messages (PublicOpenRequest, GetProcParameterRequest, PublicCloseRequest). The GetProcParameterRequest message requests the smart meter to send its current measurement values. The smart meter answers by sending a respective SML response file (see Figure 4.2a) which in turn consists of three SML messages (PublicOpenResponse, GetProcParameterResponse, PublicCloseResponse).

The GetProcParameterResponse message contains the electric power, rms-value, voltage, current and voltage-to-current phase angle for each phase as well as the phase angles between the voltages of different phases, the total electric power and the reactive power.

In the next phase of operation the answer file is parsed and a temporary measurement object is created.

The phase concerned with the server communication starts with the conversion of the Java object into a JSON representation (see Figure 4.2b) which the server understands. Then the JSON encoded measurement is sent to the server via a HTTP POST message. In the standard case it gets acknowledged by an HTTP status code 201[4].

In the last phase the parser waits for the remaining time so that one loop lasts exactly one second in total.

4.2.5 Implementation details

The parser is implemented in Java which fulfills the previously mentioned requirement of portability. It can be executed on any computer independently of the operating system as long as it is provided with a Java Virtual Machine (JVM).

Parsing a SML response file and constructing a one-to-one representation in Java objects makes use of an average of 100 kB memory and takes about 2 ms while the Java pro-

[4] HTTP status code *201 Created*: The request has been fulfilled and resulted in a new resource being created [11].

cess in total uses no more than 14 MB at a time. These values were averaged over one million parser iterations executed on a computer equipped with a 2.66 GHz dual core CPU and 4 GB RAM.

4.2.6 Push vs. pull

In principle there are two ways to obtain the current consumption data. One is to let the smart meter push it to the parser and the other is to request it regularly. The choice between these two options seems easy: Passively waiting for the smart meter to push new data exactly at the time of measurement would reduce the time delay between data acquisition and display to the user. It would also make the requirement to measure exactly every second obsolete. However, the 8-tuple sent by the smart meter does not include information on the current load. It includes only the electricity consumption in kilowatthours represented with three decimal places. This would lead to an accuracy of at most 3600 watt which is not sufficient for our purpose. As a result pulling the consumption data regularly remains the only feasible option to achieve accurate feedback. It provides a resolution of one watt and below.

4.2.7 Problems and solutions

The exact timing requirement led to an unexpected problem as mentioned earlier. Due to an unresolved bug in Sun's JVM (through version 1.6) the system clock is pushed into the future faster than real-time. This happens under certain conditions when a thread sleeps. It results in periodic changes to the Windows system clock and leads to duplicate measurements over a time span of about one minute multiple times a day. This bug has been submitted in June 2006 and is classified as "cause known" [39]. A preliminary fix is to create a daemon thread at the start of the application which sleeps for a very long time which is not a multiple of 10 ms (about 292 million years when using the maximum value for the argument of Thread.sleep()). "This will set the timer period to be 1 ms for the duration of that sleep–which in this case is the lifetime of the VM" [39]. The problem affects Windows only, and seems to be caused by the underlying implementation of clocks and timers. However, the fix resolved the problem and enables a regular measurement of the consumption values.

A first approach to the eMeter system used inline consumption sensors for single devices as data sources instead of the smart meter. These sensors use Bluetooth[5] for communication purposes and provide the possibility to remotely control devices. The coverage of single appliance power consumption is desirable since it leads to more explicit information for the user. However, this solution had also some major drawbacks. Most importantly the Bluetooth version of the sensors could hardly provide real-time feedback. A gateway is needed to gather the current load of the sensors one after the other. The more sensors are used the longer this takes–several seconds already for one device. Furthermore, gateways have to be deployed to every room since Bluetooth achieves its maximum range of about 100 m only when obstacles like walls and ceilings are absent. Another problem were frequent communication errors which required new Bluetooth discovery phases every so often. Furthermore three out of six devices stopped working without an evident reason. Although a ZigBee version of these sensors promised better results we decided to use a smart meter instead. Amongst other advantages this proved to be more reliable and realistic for a broad deployment.

[5] Bluetooth is an open wireless protocol for exchanging data over short distances from fixed and mobile devices, creating personal area networks (PANs) [46].

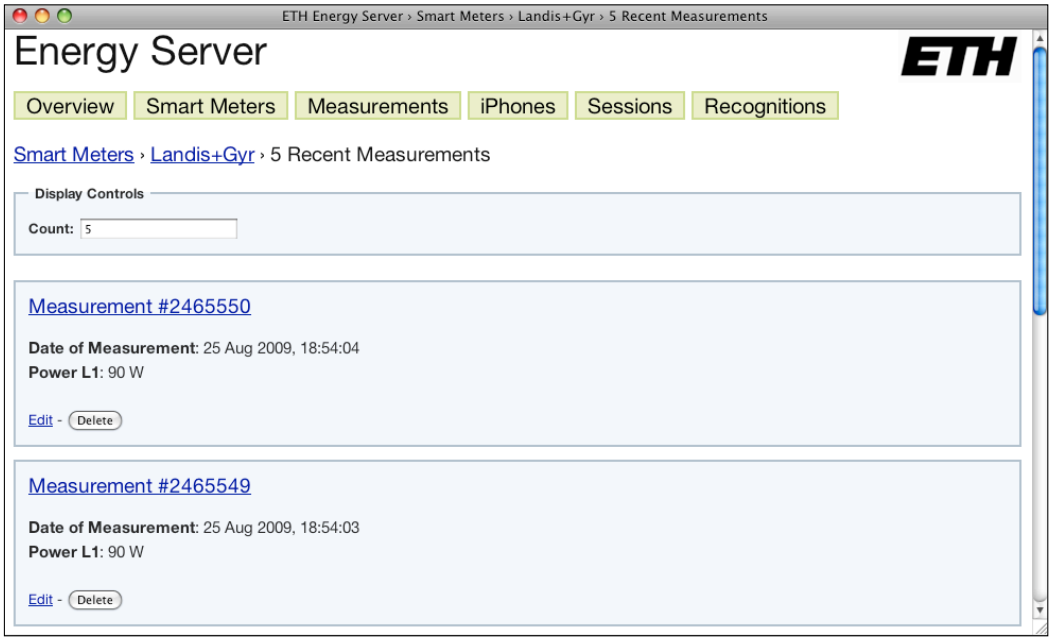


Figure 4.4: HTML representation of the five recent measurements.

4.3 Server

In times of scarce IPv4 addresses NAT[6] is commonly used as an interim solution. However, NAT breaks the originally envisioned model of IP end-to-end connectivity across the Internet. This means that computers in an internal network behind a NAT device are not reachable from the Internet. The iPhone application could for instance not reach the SML parser to get the consumption data if the parser was hidden behind a NAT and vice versa. This is one of the reasons why the eMeter system needs a server which is located in the Internet. It mediates between the parser and the iPhone application by providing a known point in the middle which can be reached by both clients.

[6]Network address translation (NAT) allows the use of private IP addresses in internal networks behind routers with a single public IP address accessible from the public Internet. The internal network devices communicate with hosts on the Internet by changing the source address of outgoing requests to that of the NAT device. The NAT device in turn relays replies back to the originating device by consulting a special translation table [49].

4.3.1 Functional requirements

Primarily the server has to satisfy the requirements of the parser and the iPhone application. It has to store measurements and make it easy to retrieve them later. There is also a number of other important features: It has to manage multiple smart meters and make it possible to access aggregated data like the consumption totaled over a specific time period. The server should provide access to the stored data through a human-friendly HTML interface which provides convenience controls to present the data over multiple pages or select different time periods. Additionally, a lightweight interface is needed for machines which allows the use of HTTP and JSON to interact with the resources managed by the server.

Furthermore, the server has to support the evaluation of user studies by storing usage data and has to be prepared to store data for a future device recognition feature based on the load curve.

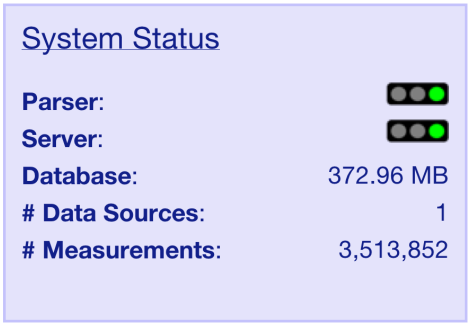


Figure 4.5: Server status information.

4.3.2 Non-functional requirements

There are also a number of nun-functional requirements which correspond to the operation of the server.

Performance: Neither the regular storage of measurements nor the request for historic consumption data should affect the performance of the server too much. Caching and preprocessing should be used to prevent the calculation over several millions of measurements.

Extensibility: Since it is very likely that new data sources will be added in the future which have to be maintained, the server should be easily extensible. Future changes should have a minimal impact on existing system functions.

Maintainability: Maintenance of the system should be made easy. The status of the system should be observable without direct access to the server. This includes that the server displays the status of the parser, its own status, information on the database like its size together with basic information on the current energy consumption. This information should be accessible using any browser.

Security: The manipulation of resources managed by the server is easily possible over HTML and HTTP. This requires that there are counter measures against intentional as well as unintentional attacks. The access to the data should for example be password protected. Additionally all inputs should be validated to prevent SQL injection attacks.

Privacy: The electricity consumption of a home tells much about the persons living in it. From the data can be deduced when they stand up in the morning, how often they wash their clothes, if they are in vacation and much more. This was recognized by Hart already in 1989. He recommends "that legal restrictions be enacted or clarified so that electric power usage is considered as private as any phone conversation" [17]. A system that stores consumption data should deal with these privacy concerns.

4.3.3 Execution environment

The eMeter server is written in PHP and can be deployed to any standard Apache server. The server stores measurements in a MySQL database which is accessed through a PDO[7] abstraction layer. The server provides status information on the status of the parser and the database as depicted in Figure 4.5 as well as short information about the current consumption and statistics about historic consumption values.

4.3.4 Recess framework

The eMeter system is built using the open source Recess framework [23]. This framework supports the development of a loosely coupled application conforming to the Model-View-Controller (MVC) design pattern. It provides useful templates for views, models and controllers.

Furthermore it provides everything needed to create a RESTful application. The Uniform Resource Identifiers are automatically routed to the corresponding methods of the controllers. It supports all HTTP methods and the transformation of model objects into a corresponding JSON representation is already built in. MVC and REST are described in the next two sections.

[7] The PHP Data Objects (PDO) extension defines a lightweight and consistent interface for accessing databases [50].

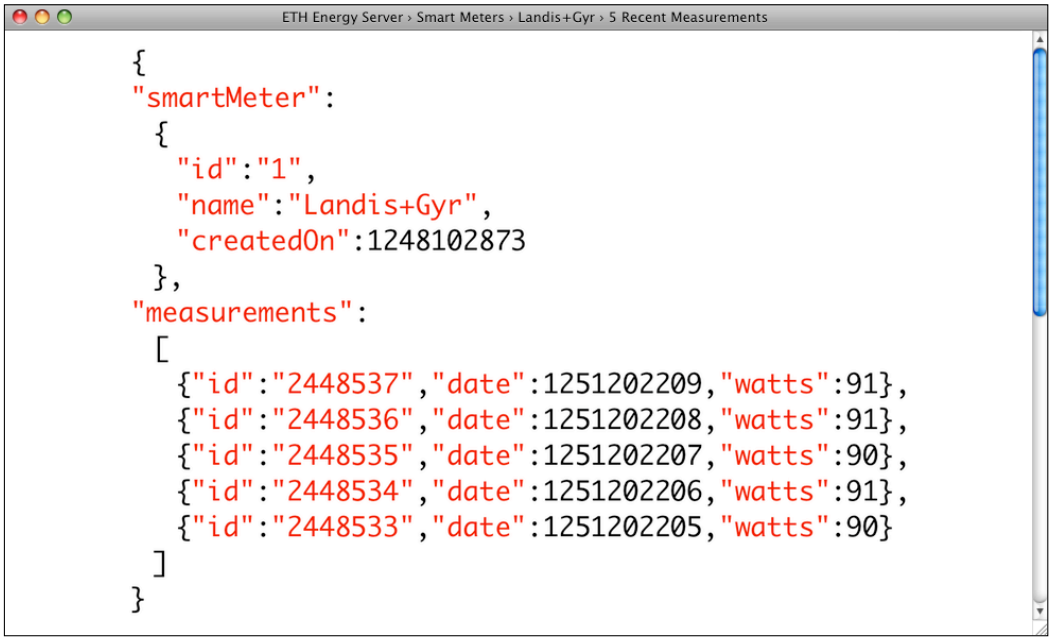


Figure 4.6: JSON representation of the five recent measurements.

4.3.5 Representational State Transfer (REST)

Representational State Transfer is an architectural style which is popular among web applications. The term was introduced by Roy Fielding in his doctoral thesis in the year 2000. He was one of the principal authors of the HTTP protocol. The following description of REST is based on [52].

What it takes to be RESTful

An application is called RESTful when it conforms to the REST principles. The largest implementation of a system which respects the REST principles is the World Wide Web. In fact REST can be seen as a post-hoc description of the architectural style of the web. However, it is not a requirement for an application to use HTTP or other web technologies to be considered RESTful. Instead a system is considered to be designed according to the REST architectural style when it conforms to set of constraints. Namely a system has to consist of clients and servers with uniform interfaces and has to be stateless, cacheable and layered. It can provide code on demand which is the only optional constraint. The next paragraphs describe these principles in detail.

Resources: The main concept of REST is that of a resource. A resource denotes a specific source of information managed by the server. This could for example be smart meters or measurements. Resources can be addressed with a uniform identifier. This would for instance be an URI in the case of HTTP. When a request which is addressed to a specific resource using this uniform identifier is sent to a server, a corresponding representation is returned to the client. Representations can be any kind of documents like HTML pages or JSON files. This representation is conceptually separate from the resource. A resource is a fictive object and representations are the concrete documents which describe the resource. A representation characterizes the current or future state of a resource and contains the actual information. It provides also all details needed to manipulate e.g. modify, create or delete a resource. A resource can be manipulated by exchang-

ing representations of it using a standardized protocol like HTTP.

Client-server: A REST architecture consists of clients and servers. These are separated by uniform interfaces. Clients send requests to servers which perform appropriate actions and answer by sending a response back to the client. Requests and responses are built around the transfer of representations to and from resources.

Stateless: Between two requests no client context is stored on the server. Each request is self-descriptive so that no further information is needed to serve it. Any state is maintained by the client. For example when a process on the client side consists of more than one request, the server does not know that the different requests are related. This makes server code easier and more reliable in the case of network errors. It also improves the scalability of the server.

Cacheable: Clients have to be able to cache frequent requests. This can eliminate some client-server interactions which further improves scalability and performance.

Uniform interface: The uniform interface between client and server decouples both as mentioned above. Servers and clients can be developed and replaced independently as long as the interface remains the same.

Layered system: Clients cannot tell if they communicate with the target server directly or with intermediary servers. This enables layered caching as well as load-balancing and such improves the performance of the system.

Code on demand: Servers can extend the functionality of a client by sending executable code to it. An example to this is Java Script. This constraint is the only optional one. All other constraints have to be met or a system cannot be described as being RESTful.

Goals

Goals of the REST architectural style are the scalability of interactions between different components, the generality of interfaces and the possibility to deploy the different components independently.

4.3.6 REST in the eMeter system

The eMeter system is built in a RESTful way. It manages the following resources: smart meters, measurements, iPhones, sessions and recognitions. A smart meter has multiple measurements and a measurement belongs to exactly one smart meter. iPhones and sessions are managed for the evaluation of user studies. I will describe them in a later section in detail. Recognitions are managed by the server as a first step towards device recognition which I will discuss later on, too.

All resources can be manipulated using the respective URIs. To create for instance a new measurement over 42 W using the command line tool curl, nothing more is needed than the following lines of code:

```
curl -X POST -d
'{
  "measurement":
  {
    "powerAllPhases":42,
    "powerL1":42,
    "powerL2":0,
    "powerL3":0,
    "currentNeutral":0,
    "currentL1":0.37,
    "currentL2":0,
    "currentL3":0,
    "voltageL1":224.6,
    "voltageL2":0,
    "voltageL3":0,
    "phaseAngleVoltageL2L1":-1,
    "phaseAngleVoltageL3L1":-1,
    "phaseAngleCurrentVoltageL1":304,
    "phaseAngleCurrentVoltageL2":-1,
    "phaseAngleCurrentVoltageL3":-1,
    "smartMeterId":1
  }
}'
http://www.example.org/emeter/
    energyServer/measurement.json >
    post.output
```

This command sends a JSON representation of the measurement to the server. The HTTP POST method is used because the resource is to be created. The request is routed

URI	HTTP Method	Description
/energyServer	GET	Index page
/energyServer/current	GET	Current consumption
/energyServer/statistics	GET	Consumption statistics
/energyServer/status	GET	Server status information
/energyServer/measurement	GET	List measurements
/energyServer/measurement	POST	Create new measurement
/energyServer/measurement/{ID}	GET	Show measurement detail
/energyServer/measurement/{ID}	PUT	Update measurement
/energyServer/measurement/{ID}	DELETE	Delete measurement
/energyServer/smartMeter	GET	List smart meters
/energyServer/smartMeter	POST	Create new smart meter
/energyServer/smartMeter/{ID}	GET	Show smart meter detail
/energyServer/smartMeter/{ID}	PUT	Update smart meter
/energyServer/smartMeter/{ID}	DELETE	Delete smart meter
/energyServer/smartMeter/{ID}/measurements	GET	List recent measurements
/energyServer/smartMeter/{ID}/kWh	GET	List kWh

Figure 4.7: Example URIs of the eMeter server.

to the respective method based on the URI "/energyServer/measurement.json".

The response from the server acknowledges the creation of a new measurement resource with an HTTP *201 Created* status code. The answer also contains the measurement's new location and its creation date:

```
HTTP/1.1 201 Created
Date: Thu, 1 Sep 2009 09:51:48 GMT
Location: http://www.example.org/
    emeter/energyServer/measurement
    /3557140

{
  "measurement":
  {
    "id":3571146,
    "powerAllPhases":42,
    "powerL1":42,
    "powerL2":0,
    "powerL3":0,
    "currentNeutral":0,
    "currentL1":0.37,
    "currentL2":0,
    "currentL3":0,
    "voltageL1":224.6,
    "voltageL2":0,
    "voltageL3":0,
    "phaseAngleVoltageL2L1":-1,
    "phaseAngleVoltageL3L1":-1,
    "phaseAngleCurrentVoltageL1":304,
    "phaseAngleCurrentVoltageL2":-1,
    "phaseAngleCurrentVoltageL3":-1,
    "createdOn":1252590360,
    "smartMeterId":1
  }
}
```

Other examples for URIs of the server and the effect of different HTTP methods when applied to them are described in Figure 4.7. This table contains a selection of URIs which can be used to request information about the general system as well as measurements and smart meters. They can also be used to manipulate them by using the correct HTTP method. GET queries a resource or a collection of resources. POST creates a new resource and returns the automatically generated ID. PUT updates a resource and is therefore executed on the specific URI of a resource. Finally DELETE removes a resource. POST, PUT and DELETE cannot be cached because they have side effects. GET requests should never have side effects and therefore can be cached.

The server also supports different media types. Figure 4.4 and 4.6 are an example to this. They depict the collection of the five recent measurements from the smart meter in different media types. Namely these are HTML and JSON representations. The

HTML representation is human readable and allows to navigate through measurements and smart meters using web links, paging and visual interfaces to select different time spans. For machines it is in the contrary easier to use JSON as it provides a well defined representation of the resources without redundant information. The server supports both file extensions like ".json" and ".html" as well as the HTTP accept header to request a specific media type. The header needs to be set to "Accept: application/json" or "Accept: text/html".

4.3.7 Model-View-Controller (MVC)

The MVC paradigm is a widely used architectural pattern that supports the creation of loosely coupled systems. It was first described by Trygve Reenskaug in 1979 and has been widely implemented ever since.

MVC separates the model which contains all the data from the view which manages the user interface. The controller mediates between the two.

Model: The model holds all the data and provides an interface to manipulate it. It encapsulates the actual storage mechanism which could for example be a database. The model is completely decoupled from the view and the controller as it knows nothing about their implementation.

View: The view displays the models data so that a user can interact with it. The view manages the user interface and its design. Actions of the user are forwarded to the controller. The view is not concerned with data storage or how to respond to user interactions. It gets the data which it displays from the model. Multiple views can exist for one model and one view can display data from multiple models.

Controller: The controller glues views and models together. It decides what to do if the user performed a specific action on the user interface. The controller notifies the model of the user action which possibly results in a change of the model's state. The controller does not know how the view looks like or how the data storage is implemented.

Advantages: The MVC paradigm decouples views and models which leads to a loosely coupled architecture. By this it reduces the complexity and increases the flexibility. This makes it easy to add or exchange visualizations and data sources without a huge impact on other parts. This also increases the possibility to reuse code.

4.3.8 MVC in the eMeter system

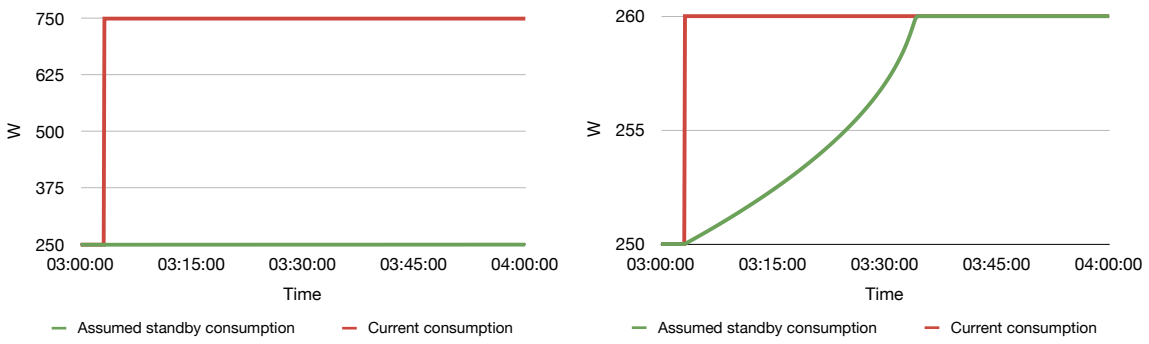
The eMeter server is built conforming to the principles of MVC. Its model abstracts from a database. Model objects are for example smart meters, measurements, iPhones, sessions and device recognitions. Views include lists of resources, resource details and forms to edit resources. Controllers manage the model objects and route URIs to the appropriate methods.

For example the SmartMeterController reacts to a GET request which is performed on the specific URI "/emeter/energyServer-/smartMeter/4/measurements" by returning a representation of the recent measurements of the smart meter with the ID 4. These measurements are then handed over to the view which displays them in a list while adding hyperlinks for possible future user interactions.

MVC made the development of the eMeter system easier by providing high flexibility. At first the server only managed smart meters and measurements. Later iPhones, sessions and recognitions were added. This was easily possible due to the use of the MVC architectural pattern.

4.3.9 User study support

The server provides support for the evaluation of future user studies. It manages mobile phones and user sessions. These describe how long the user remained in each view and how many devices were measured, saved or



(a) Party night

(b) New device

Figure 4.8: Adaption of the assumed standby consumption level in two different scenarios.

updated. In this case the source for this data is the iPhone which sends the data every time the application is terminated. Of course this logging can be disabled in the settings of the iPhone application.

4.3.10 Device recognition support

Besides the support for user studies the server also has basic functionalities to enable future device recognition. Every time the user measures a device the corresponding part of the load curve is tagged and could later be used to extract device characteristics.

4.3.11 Standby consumption

The iPhone application uses colors to provide context information for the range of the current consumption. Blue stands for the standby level and green, yellow and red are ratings based on the specific household. This allows users to see with a quick look if the consumption of their home is in the standby range when leaving it. This would mean that everything is shut down and they need not to worry whether some appliances are still turned on.

However, there is no simple way to conclude from the load curve on the standby level. One solution would be to take the average over the consumption between three and four o'clock of the last night–assuming that at night every device is in standby mode. However, this does for example not apply if the inhabitants had a party last night. So how about an average over the nightly consumption over all days? This seems to compute the level of standby consumption unaffected by infrequent parties but it has an apparent draw back: If the standby consumption of the home changes due to new appliances like a new TV it would take a long time for the average to reflect this change. A calculation over the nightly consumption of the last month would solve this problem since it would adjust to changes after a month at the latest but this may require extensive calculations while still not adapting very fast.

I thought of another solution: Consumption values measured between three and four o'clock are a priori thought to reflect the current standby consumption. A weighted average over each new value measured in this time frame and the previously assumed level of standby consumption is calculated. This is then assumed to be the new standby consumption. The weight for the new measurement corresponds inversely proportional to the difference between it and the previously assumed standby consumption. With the result that phases of high consumption like parties have little influence (Figure 4.8a) while small changes caused for example by a new device lead to a fast adaption (Figure 4.8b).

This approach assumes that the standby consumption is not increased or decreased to a large extent.

4.3.12 Problems and solutions

A first prototype for the eMeter server was developed using GlassFish. This is an open source application server project led by Sun Microsystems for the Java EE platform. The application was written using the Java API for RESTful Web Services (JAX-RS). JAX-RS uses annotations to support the creation of web services according to the REST architectural style. Derby was used as a database for the measurements. This solution allowed to built a running prototype quickly. However later on it proved to be unusable for a permanent deployment. The GlassFish application server was highly oversized for the purposes of the eMeter system and needed too many system resources. The Recess approach in contrast is more lightweight, provides templates for the user interface and is better customizable. The MySQL database can be easily managed with standard tools like phpMyAdmin.

A problem did become increasingly apparent after about a week of operation when the database held already over 300,000 measurements. The web server used about 104 % of the CPU time (two CPUs) only for the eMeter application. The reason for this was very simple. The server loaded by default all previous measurements from the database when a new measurement was created. I replaced the responsible code with custom one. The load did go down to 0–1 % where it remained until today with over 3.5 million measurements in the database. Another performance issue occurred each time the eMeter iPhone application requested aggregated consumption data. This was resolved by maintaining cache tables for predefined time spans which are updated every time a new measurement object is created.

4.4 iPhone application

The front-end of the eMeter system is provided via the iPhone application which I will describe in this section.

The user interface is the most important part of the eMeter system since when it fails to provide useful feedback the whole system becomes redundant. It is the part with closest user contact so it should be designed very carefully.

4.4.1 Functional requirements

The iPhone application should provide the users with the functionalities as described by the UML use case diagram depicted diagram in Figure 4.9.

Overview

The user should have the possibility to get an overview of the level of current consumption in his home. This display should adjust automatically when the consumption exceeds the limit of the display.

History

Historic comparisons are useful to provide additional feedback. Therefore the iPhone application should provide a view which displays the historic consumption with bar and line graphs as well as energy statistics. It should be possible for the user to select different time periods since this was recommended by several user studies as described in Section 3.1.3. The view should express historic consumption in different metrics like cost and energy consumption. This can help users understand the data better (see Section 3.1.5).

Measure

An appliance specific breakdown of the power usage allows for a much higher involvement of the user (see Section 3.1.2). Since the eMeter system has no appliance specific sensors the

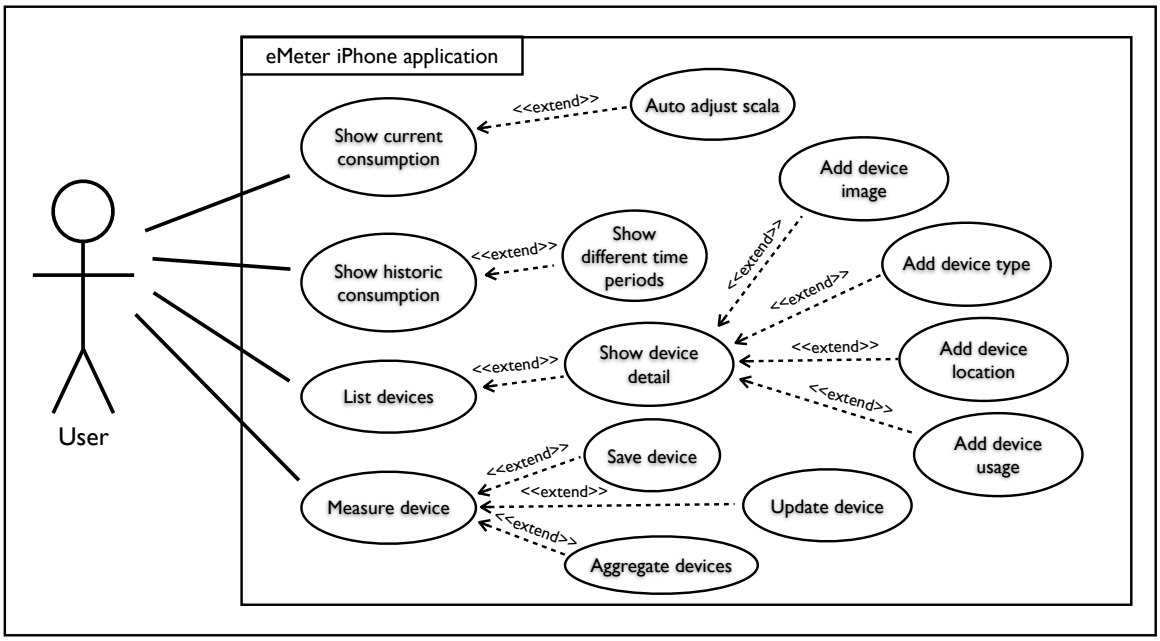


Figure 4.9: UML use case diagram describing the functional requirements.

iPhone application should provide a solution based on the load curve. It should enable users to measure a single device and save the measurement as a device or update the consumption of a previously saved device.

Devices

This view should display a list of all saved devices. Users should be able to select a device for which they want to see detailed information. It should be possible for them to add a custom photo to the device, categorize it by providing type and location or adjust a simple usage plan which should be used for extrapolating cost and consumption.

4.4.2 Non-functional requirements

The most important quality requirement for the iPhone application is a high standard of usability.

Usability

It is important that the visualization of the current load is easy to understand since it is the primary feedback for the user about his energy consumption. While it should be clear at a first look it should also provide more information than just the level of current consumption. The user has to be provided with comparisons to understand the bare numbers.

The view which enables the measurement of single appliances should be built to be fun to use. Users should interact with their devices in a playful way. They should not wait a long time to get the result of a measurement. They should have the possibility to personalize their devices by adding photos which make it easier to tell different devices apart.

Responsiveness

The other important requirement is the responsiveness of the user interface. Since the application has to perform network communication activity constantly this has to be built in a way which does not affect the responsiveness of the user interface. Every animation, network communication and expensive calculation should be implemented as a background task.

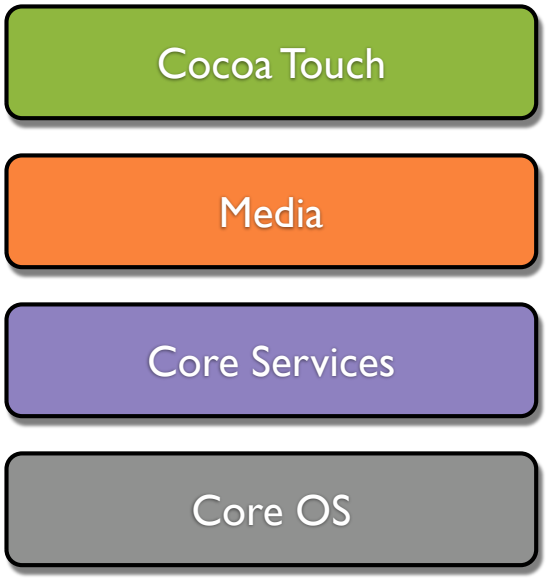


Figure 4.10: Service layers of the iPhone OS.

4.4.3 Software platform

In this section I will describe the characteristics of the iPhone as a software platform. It is very similar to the Mac platform since it uses an adapted version of OS X as its operating system. I will first discuss the system architecture, then the tools used to built an iPhone application and the programming language Objective C. At last I will discuss specifics of iPhone applications.

System overview

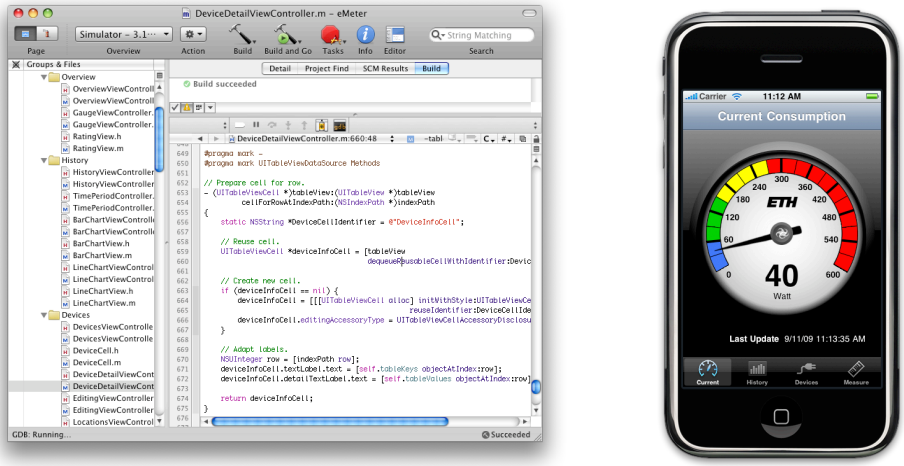
iPhone OS is based on a variant of the Mach microkernel[8]. On top of this are the layers of services that provide the infrastructure used to implement applications for the iPhone. I will describe each of them from bottom upwards by roughly following the description in [3]. These four layers, namely Core OS, Core Services, Media and Cocoa Touch, are depicted in Figure 4.10.

Core OS and Core Services: Core OS and Core Service are the fundamental basis of the iPhone OS. They provide interfaces to low-level functionality including file storage and access, low level data types, POSIX threads, BSD sockets. These interfaces are mostly written in C. However, higher layers of the OS provide abstractions of these fundamental services which are more convenient to use. Most of the times the functionality of these layers will not be used directly. Still, if low-level access is needed, it is possible to access these fundamental services directly. A reason for this could be the implementation of custom behavior that is not provided at a higher level.

Media: The Media layer provides interfaces which handle audio, video as well as 2D and 3D drawing. It offers access to technologies like OpenGL ES, Quartz, Core Audio and Core Animation. OpenGL ES provides 3D graphics, Quartz manages 2D drawing, Core Audio handles audio playback and Core Animation is an advanced animation engine. The APIs of the Media layer are a mixture of C-based and Objective-C interfaces. This layer would be used for example when playing videos or rendering 3D graphics.

Cocoa Touch: The Cocoa Touch layer provides high-level abstractions to almost all underlying technologies. Its APIs are the ones used most of the time when developing a typical iPhone application. It consists of several frameworks which encapsulate related functionality. The Foundation framework for example provides object-oriented support for network access, file management and object collections. The UIKit framework handles everything concerning the creation of user interfaces. It manages view hierarchies and provides classes for windows, views, controls as well as controllers that manage these objects. Other frameworks allow the access to the users photos and contacts, to location and accelerometer data as well as the ability to handle multi-touch gestures.

[8] Mach is an operating system microkernel developed at Carnegie Mellon University to support operating system research, primarily distributed and parallel computation. It is one of the earliest examples of a microkernel, and still the standard by which similar projects are measured [48].



(a) XCode IDE (b) Simulator

Figure 4.11: Tools used to develop iPhone applications included in the iPhone SDK.

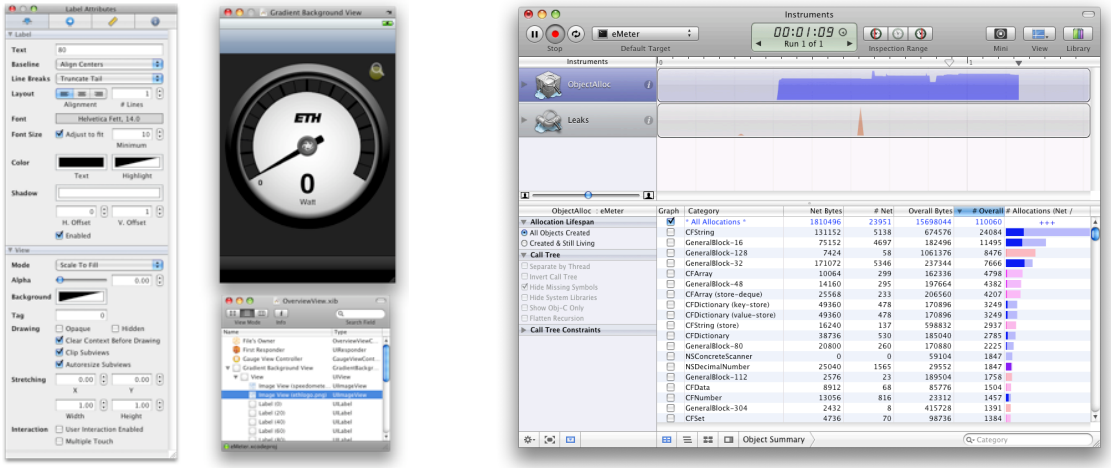
Tools

XCode is a suite of development tools which originates from the Mac development and has been around for years and is now also used to built iPhone applications. It provides support for project management, code editing, building executables, source-level debugging, source-code repository management, performance tuning, and much more. XCode is available for free download as a part of the iPhone SDK. This includes the XCode IDE, an iPhone simulator, the Interface Builder and Instruments. They are displayed in Figure 4.11 and Figure 4.12.

XCode IDE: The XCode IDE is the tool of the iPhone SDK which is used most of the time. It is the center of the development process since it is the source-code editing environment and the starting point for the other tools. It is an integrated development environment (IDE) like Eclipse or Netbeans. As such it provides source code editing, building of executables and debugging support for both the real iPhone device and the Simulator. The XCode IDE manages all files of an iPhone project including resources like images, videos, interface files and the source code of course. Additionally it provides the access to API documentations. The XCode IDE is depicted in Figure 4.11a.

iPhone Simulator: The iPhone simulator provides a local environment for testing applications on the same computer used for development. It is suited to test new functionality quickly and to make sure the application behaves as intended. However, the Simulator is not an emulator which means that there are certain differences to an actual iPhone. It is not an adequate tool for performance testing since the Simulator executes code which is compiled for the computers x86 CPU. This leverages all the performance provided by the computer which is far more than that of an iPhone. So it is still important to test on real devices, too. An actual iPhone is needed mostly for testing the user interaction on a touch screen and for discovering performance issues. A screenshot of the iPhone Simulator running the eMeter application is displayed in Figure 4.11b.

Interface Builder: Interface Builder is a tool to assemble user interfaces visually. It provides standard components like buttons, labels, text fields and switches as well as custom views. These components can be added to the main window using drag and drop.



(a) Interface Builder

(b) Instruments

Figure 4.12: Tools used to develop iPhone applications included in the iPhone SDK.

While arranging them, reference lines appear which help to conform to the iPhone Human Interface Guidelines (HIG). An inspector is used to change certain attributes of the components and connect them to the source code. The user interface can be designed completely in the source code never starting Interface Builder. However, the visual arrangement leads to a better separation of code and interface which makes it easy to change it independently from the code. Interface Builder allows it to see already at design time how the interface will finally look in the application which is not possible when specifying it solely in the source code. Interface Builder consisting of the inspector, the document window and the user interface of an application window is depicted in Figure 4.12a.

Instruments: Instruments allows to analyze the performance of iPhone applications running in the Simulator or on an actual device. It displays the performance data, which is gathered live from the application, visually in a timeline. Instruments allows to look closely at memory usage, disk activity, network activity and graphics performance. Simultaneously testing specific functionalities and keeping an eye on the live analysis conducted by Instruments, allows to discover hot spots, bottle necks and memory leaks in the application code. A session of Instruments which found a memory leak can be seen in Figure 4.12b.

Objective C

Objective C is the main programming language used when developing iPhone applications. It is also the primary language used by Apple's Cocoa API. Objective C is an object oriented programming language which extends the standard ANSI C language with a syntax for defining classes and methods. It derives its object syntax from Smalltalk. Al the syntax for non-object-oriented operations is identical to that of C, while the syntax for object-oriented features is an implementation of Smalltalk-style messaging.

The most obvious difference to other programming languages is the way methods are called. For example in Java the call to an object would look like:

```
object.method(parameter);
```

which would translate into the following Objective C code:

```
[object method:parameter];
```

In Objective C calling methods is much more like sending messages to objects. This is reflected in the name of methods which can be almost read as prose. This makes it easy to tell what the different arguments describe. A good example to illustrate this is:

```
[finder openFile:mailing
    withApplication:@"MailDrop"
    andDeactivate:YES];
```

On the contrary in Java it is not clear what the single attributes stand for:

```
finder.openFile(mailing, "MailDrop",
    true);
```

A major difference between Objective C on the iPhone and Objective C on a Mac is the absence of garbage collection. All memory allocations have to be released manually.

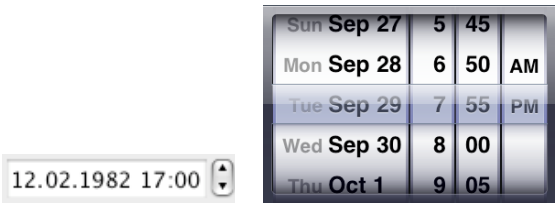
Apart from that, Objective C is very similar to other object-oriented languages. Further details on concepts like class definition, memory management, protocols and properties are available in the documentation at the iPhone Developer Center [2].

Specifics of iPhone applications

There are some major differences between an iPhone and a desktop computer. These differences have to be kept in mind while designing an application for the iPhone. Major issues will be discussed in this section.

Users are mobile: First users of a mobile phone application are most likely on the go. As a consequence they do not have time to interact with complex interfaces. Instead they want to get the desired information easy and fast. If it takes too long they will simply quit the application. When the application repeatedly fails to meet expectations it will be removed from the phone.

Restricted screen size: An iPhone has indeed a large screen when compared to other mobile phones but it is fairly small in comparison with standard computer screens. For this reason applications cannot have several windows to display content. There is simply just one window with a size of 480 × 320 pixels. However this restricted screen size can also act as a motivation to focus on essentials as opposed to making the user interface unattractive by crowding it with elements. Applications should be designed to present only screenfuls of content at a time. This leads to the development of drill down menus and elaborate view hierarchies. They have to be designed carefully to remain understandable.



(a) Date picker Mac (b) Date picker iPhone

Figure 4.13: Differences between platforms.

Input methods: Another rather apparent difference is the absence of mouse and keyboard as input devices. Instead the users interact with their fingers. Interface elements have to be big enough so users can easily hit them. A demonstrative example for this is displayed in Figure 4.13. The depicted date pickers have been optimized for the use with a mouse and a finger respectively. The iPhone uses a virtual keyboard in place of a real one. Certainly much work has been spent designing it, but text input remains tedious and is for example impossible while the user is walking. As a result text input should be avoided when possible.

Memory limitations: Memory is not unlimited on the iPhone and as disk swapping is not available memory management becomes even more important. An application should respond to memory warnings by cleaning up unused memory. An application can be designed to have a small memory footprint by eliminating memory leaks, reducing the size of resources and loading resources not until the time when they are used.

One application at a time: Due to the limitation of resources iPhone applications cannot run in the background. Switching between application results in terminating the one currently running and launching another. Even answering the phone quits the running application and restarts it later. As a consequence the launch time of an application is critical and has to be optimized vigorously. Furthermore this means that an application is quit without much warning. User changes should be saved at the time they are made and restored the next time the application is launched. At best the user does not realize the application was quit.

4.4.4 User interface fundamentals

In this section I will describe all issues concerning the design of the user interface for the eMeter iPhone application in detail.

Conforming to the iPhone HIG

Apple provides iPhone developers with detailed recommendations how user interfaces for the iPhone should behave to fit into the iPhone application environment. These principles are recorded in the iPhone Human Interface Guidelines (HIG) which are also available at the iPhone Developer Center [2]. Besides descriptions of how to use the standard controls it also contains universal principles which I will discuss briefly in this section.

Metaphors: When possible application objects and actions should be modeled on objects and actions of the real world. This can help novice users of the application to understand quickly how to interact with it. It is important that new applications are consistent with metaphors that exist in the iPhone OS. This helps to be consistent with other applications and with the expectations of experienced iPhone users. Metaphors should be used only if the majority of the users know them and if they are appropriate. Otherwise they will most probably confuse users.

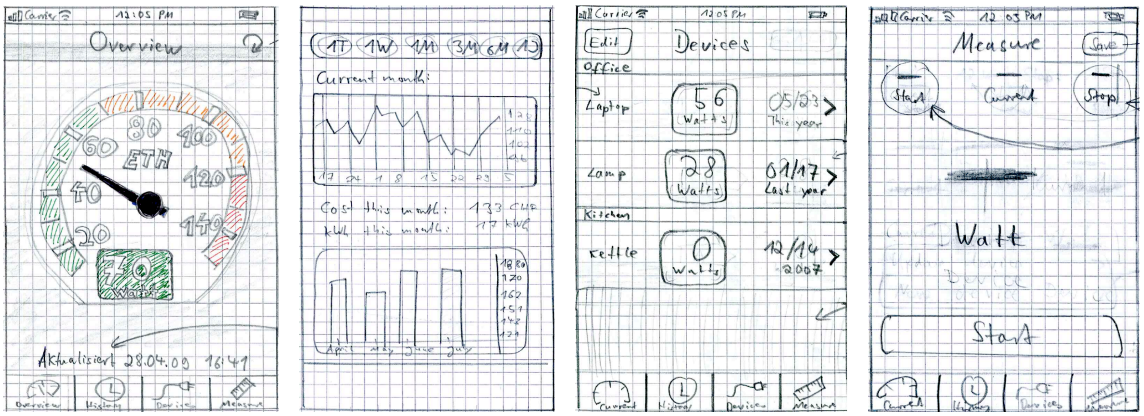
Direct manipulation: Direct manipulation means that users interact directly with objects and see the results of their actions immediately. iPhone applications can take direct manipulation to another level since users interact with controls using their fingers as opposed to the more indirect manipulation with a mouse. Natural gestures like pinching for zooming can increase the sense of direct control over the displayed objects.

Provide lists of options: An application should provide lists of options instead of asking the user to input text. This saves time for the user because typing takes longer than just scrolling a list and tapping on a row. It also reduces the need for error checking on the application side.

Feedback: Applications should respond to all user actions with some visible feedback. Otherwise the user would be uncertain if his interaction was recognized by the application. Without feedback users will most probably use controls multiple times which at worst results in undoing things again. During operations which take several seconds like network communication the application should show an activity or better a progress indicator. Animation can enhance user experience as long as it is provided both subtle and meaningful.

User control: The user should be in control of actions and not the application. It should be possible for the user to cancel operations before they even begin or stop them while they are running. In the case an operation would be destructive the user should be asked for confirmation.

Aesthetic integrity: The appearance of an application is as important as the functionality it provides. If the interface is cluttered the application becomes hard to understand and operate. However aesthetic integrity is not a measure of how beautiful an application is, but on how well the appearance fits the functionality. The benchmark for this can vary significantly between different types of applications for instance when comparing games to productivity applications.



(a) Overview tab (b) History tab (c) Devices tab (d) Measure tab

Figure 4.14: Paper prototypes for the main view of each tab.

Paper prototyping

Paper prototyping is a method of usability engineering which can provide useful feedback at an early stage of the design process. This will most probably result in better user interfaces since problems are revealed early– even before work has gone into the actual interface design with Interface Builder which is much more time-consuming.

For the design of the iPhone application paper prototypes were primarily used to identify how much information fits well into one screen and how the controls should be assigned to different views. Paper prototypes were drawn in multiple iterations for every tab and almost every view of the iPhone application. A selection of them is depicted in Figure 4.14. Displayed are paper prototypes for each main view of the overview, history, devices and measure tab.

4.4.5 User interface discussion

The eMeter iPhone application is divided into four main screens which correspond to the four functional requirements. The user can switch between them using a tab bar at the bottom of the screen. In this section I will describe each of the four tabs in detail.

Overview tab

The Overview tab displays the current consumption of the home. As visible in Figure 4.15 the view is geared to look like a speedometer. I have chosen this metaphor since it is well known from cars, airplanes and other vehicles. It allows to grasp the range of the consumption with a quick glance.

Besides its simplicity there were quite a few issues to solve. Since the communication with the server is based on an Internet connection, network problems had to be taken into account. The application tries to update the current consumption every second and refreshes the time of the last update displayed at the bottom (see for example Figure 4.15d). If a connection is not possible the application stops updating and waits for the connection to be reestablished. Previously the user had to press a dedicated button to connect to the server again (see top right of Figure 4.15b).

The scale is colored in blue, green, yellow and red which provides the user with context information to understand the current load of his home better. Blue is the range of standby consumption which is determined by the server as described in Section 4.3.11. The green, yellow and red ranges are ratings based on the consumption of the home. They are calculated based on the maximum consump-

(a) First version (b) Second version (c) Final version (d) Scale zoom enabled

Figure 4.15: Evolution of the overview tab of the eMeter iPhone application.

tion or the median. One of the two possibilities can be chosen by the user in the settings of the application.

The power usage of a home has no defined limit which is a difference compared to the speed of a car. As a result the speedometer-like visualization has to be able to increase its scale when the consumption exceeds the upper limit. It was first implemented in a way that the scale zoomed to the appropriate range automatically and was also scaled down when the current load fell under 25 % of the maximum. This provided the user with as much detail as possible. However usability tests have shown that it confused novice users and needed to be explained. In the final version the following two modes were introduced instead: In the standard mode the scale limit is the consumption maximum which is provided by the server. This means the scale will be increased when the limit is exceeded but is never scaled down. This surely has the disadvantage that small consumption levels are not displayed in full detail. What is more important, it gives the user the possibility to understand how much electricity his home consumes without checking the scale first. In the second mode the scale is also scaled down in case the consumption fell under 25 % of the current scale limit.

The user can switch between both modes by double tapping the speedometer. However, in the usability test users double tapped a second time because they thought their first double tap was not recognized since zooming takes some time. This cancelled the effect of the first double tap because it deactivated the zooming again. Apparently feedback about the ongoing operation was missing as described in Section 4.4.4. The activation and deactivation of zooming is now visualized with a magnifier glass icon which flashes while the scale is adjusted and stays if the zoom is enabled or hides away when it is disabled. This icon is depicted in the top right corner of Figure 4.15d. Additionally animation was added to point out when the scale changes its size. Labels are faded out, the color ranges move accordingly and at last the labels fade in again displaying the new values.

The final design of the speedometer evolved from several iterations. The first version was to dark to be easily readable as visible in Figure 4.15a. The second version was still too dark and additionally users associated the pitch lines of the scale with shark teeth (Figure 4.15b). The final version was approved by the users to be both readable as well as nice to look at (Figure 4.15c).

(a) Stocks application (b) Current hour (c) Current day (d) Current week

Figure 4.16: History tab of the eMeter iPhone application.

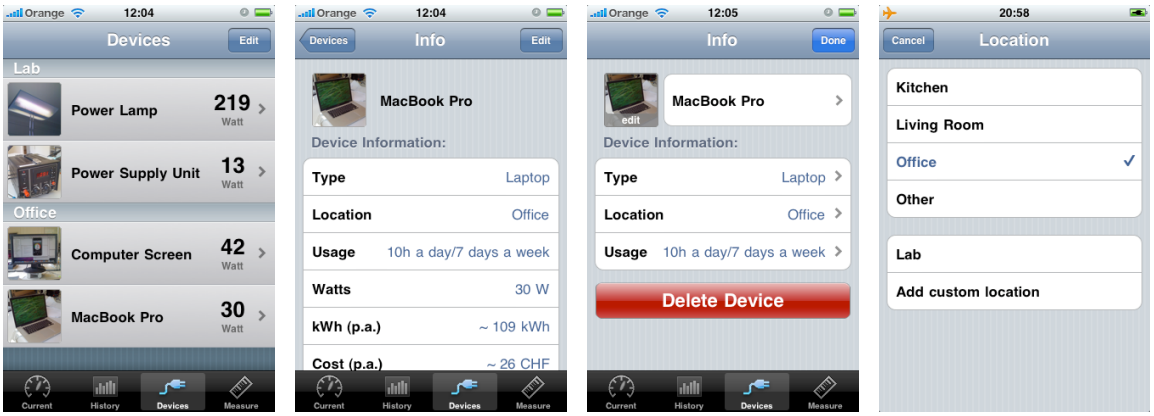
History tab

The history tab displays historic consumption values. The standard stocks application was used as a template since it also displays graphs over different time spans. The time selection control for example was built to look and feel like the one of the stocks application (compare Figure 4.16a and 4.16b). The buttons use the same abbreviations and glow in the same way when selected. The selection also moves from button to button when the finger moves from left to right and the other way around. This ensures that users which are accustomed to the stocks application are not confused when using the eMeter application but rather now already how to use it.

The history tab consists of a line graph at the top and a scroll view at the bottom. The scroll view has three sub-views. The first displays statistics about the selected time span, the other two display each a bar graph indicating the consumption and respectively cost over the last five time spans. The time selection view at the top allows the user to choose between the following time spans: hour, day, week, month, quarter, half-year and year. A bar graph displaying consumption over the last five hours is displayed in Figure 4.16b. The other bar graph depicting cost (in this case over the last five days) is shown in Figure 4.16c. The statistics for the current week including the amount of electricity used, the cost, the maximum and average watts are visible in Figure 4.16d.

The bars of the bar graph are drawn using color gradients which add additional context to the power consumption. The rating is based on the number of tenants and square meters of the home. Both can be specified by the user in the settings. The reference consumption values used for this rating originate from the consumer association Nordrhein-Westfalen but can easily be replaced with others or even requested from the server which could manage appropriate values for different countries and regions.

The user can shake the device or reselect the selected time period to reload the consumption data. An activity indicator is displayed while data is being loaded from the network and the user interface remains responsive for user interactions. This is accomplished by using background threads for the network communication. Consumption data from previously selected time periods is cached for the time the application is running. When the user selects a different time period the cached data is displayed first and gets refreshed at the time new data arrives from the network.



(a) List of devices (b) Device detail (c) Edit device (d) Add location

Figure 4.17: Devices tab of the eMeter iPhone application.

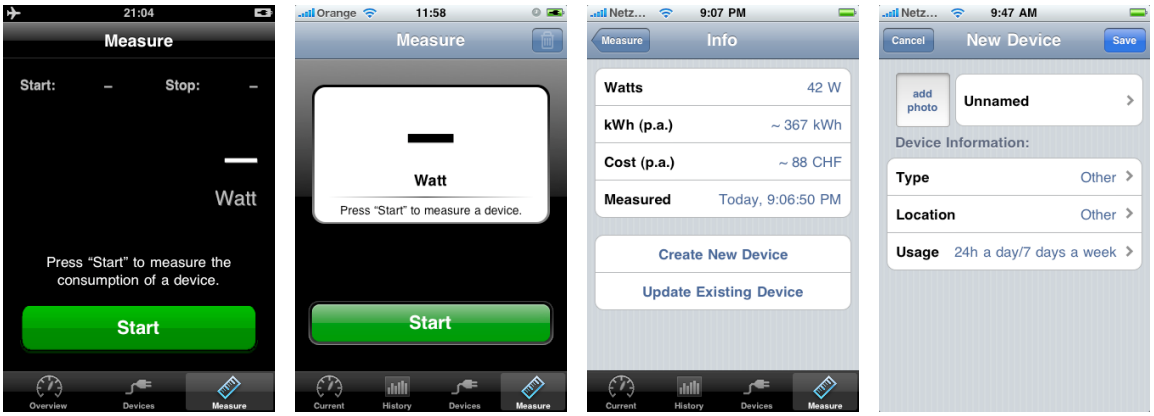
Devices tab

The devices tab lists all measured devices as depicted in Figure 4.17a. They are grouped by location and ordered by name. The list displays a picture, the name and the consumption for each device. Single devices can be removed from the list with the strike out gesture which is a common metaphor on the iPhone. The user has to confirm this action since it is destructive as a device cannot be restored. A standard iPhone list has white cells and a white background. The list of the eMeter application was designed to look like the one of the standard iPhone world clock application. The cells have a custom background gradient and shadows. This creates an unobtrusive 3D look. A background image is visible behind the list. If the list is empty a label with the text "No devices" appears which is common for lists on the iPhone.

A device can be selected by tapping on it. As a result the device detail view slides in which displays type, location, usage, watts, consumption and cost information as well as the time the device was measured (see Figure 4.17b). This view has been designed to match the standard contact application of the iPhone. The per year consumption and cost of the device are extrapolated with respect to the usage plan defined by the user. A back button in the navigation bar allows to navigate back to the list. A tap on the photo preview shows it in full resolution. The image can be zoomed with the standard pinch zoom gesture. The edit button allows for a direct manipulation of the device. When transitioning into editing mode the list animates so that only editable attributes remain visible and an overlay over the preview image is displayed (see Figure 4.17c). Name, type, location, usage and image are editable. When the user selects an attribute a corresponding editing view is presented modally on top of the tab bar. The options for the location are given as of a list because this reduces the need for text input as recommended by the guidelines in Section 4.4.4. A few standard options are provided at the top and custom locations which are already used for other devices are displayed below (see Figure 4.17d). The user can also add additional custom locations.

Measure tab

A first version of the measurement tab had labels dedicated for the current consumption, the consumption at the start and the end of the measurement as depicted in Figure 4.18a. However, users were only interested in the result so the before mentioned labels were omitted as visible in Figure 4.18b. The new design



(a) First version (b) Final version (c) Measurement detail (d) New device dialog

Figure 4.18: Measure tab of the eMeter iPhone application.

was also said to be better readable because it is not as dark as the first. This view was designed to match the style of the stop watch application of the iPhone.

After a device has been measured its consumption is displayed. Measurement details can be inspected in the measurement detail view which is shown in Figure 4.18c. This view provides specifics on consumption and cost per year as well as the watts and the date of the measurement. Two buttons are available to either create a new device or update an existing one. Figure 4.18d depicts the modally presented view which is displayed when the user decides to save the measurement as a new device.

Device measurement

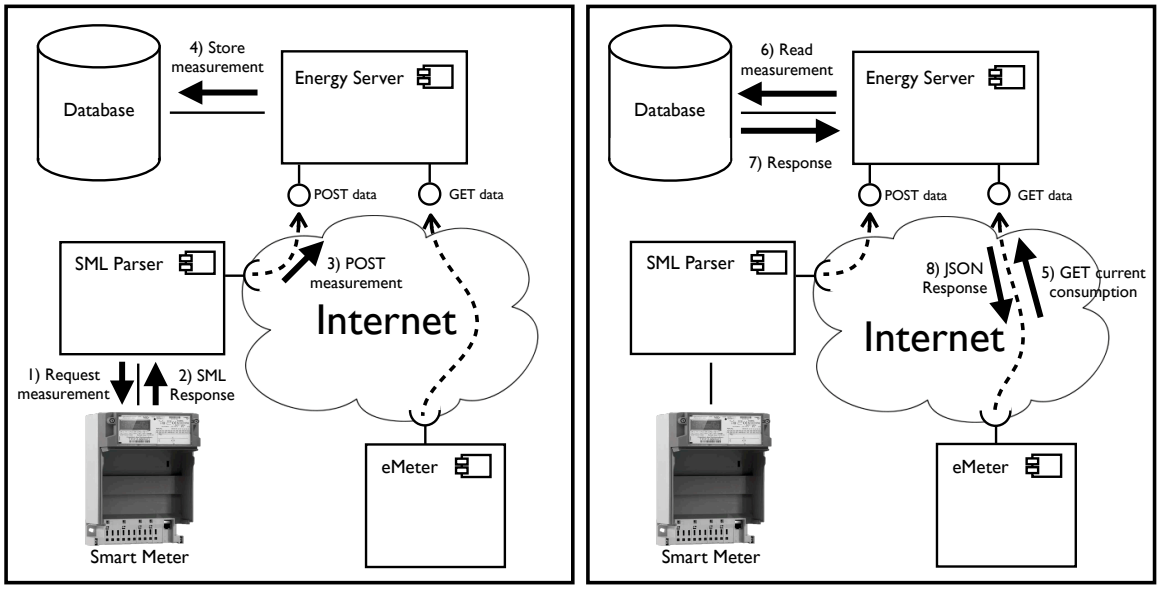
The measurement algorithm works as follows: When the user presses the start button the current load value is saved. Next the user turns the device on or off. Now the algorithm waits for the load curve to settle–two consecutive values have to be roughly the same–because some devices take several seconds to reach their final consumption level. Then the result is displayed. Although this algorithm is fairly simple it provides good results as evaluated with a reference electricity meter.

4.4.6 Problems and solutions

In a first version of the application the iPhone would become very hot after some time in standby mode while the application was still running. This was caused by the application continuously updating the consumption every second. The network communication made constant use of the Wifi module which needs energy and dissipates heat. This problem was fixed by disabling updates at the time the device is locked by the user and enabling updates again when it is unlocked.

Another optimization which saves energy is the observation of network changes. Instead of trying to connect even when the server is not reachable the application registers for notifications on the network status and updates only when the network is available.

Lazy loading was introduced to speed up the application start. At launch time the view is displayed which was selected the last time the application was used. With the intended result that the user can continue exactly where he had left off the last time. Only the required view is loaded into the memory and all other objects including detail views and model objects are not created until they are used. This saves time and memory and allows the user to switch between different applications quickly.



(a) Data acquisition and storage

(b) Data request

Figure 4.19: UML communication diagrams illustrating the main flow of information.

4.5 Communication details

This section describes how the different components fit together which have so far been described only individually.

For this purpose two UML communication diagrams are used which are depicted in Figure 4.19. They illustrate how the different components of the eMeter system communicate with each other.

With the help of these diagrams I will describe the two main communication events of the system. First the data acquisition from the smart meter and storage on the server and second the request for data by the iPhone application.

4.5.1 Data storage

The first of the two main communication processes handles data acquisition and storage. It has four basic steps as illustrated in Figure 4.19a. They recur every second.

Step 1

Step one starts with the parser opening a socket to the smart meter. It sends a special SML file over the newly opened TCP/IP connection to request the current measurement values. The SML files are described in detail in Section 4.2.4.

Step 2

The smart sends an appropriate answer as the response to the request. It contains detailed information on the current power consumption and other information.

Step 3

The parser parses the SML response, builds a JSON representation and sends it to the server over an Internet connection using the HTTP protocol. For this the RESTful API provided by the server is used.

Step 4

The server parses the JSON message and creates a corresponding measurement object as requested. The server stores the measurement in its database. The new measurement can also trigger other actions like the correction of the estimated standby value as described in Section 4.3.11.

4.5.2 Data request

The other major communication event in the eMeter system is the request of consumption data by the iPhone application. This happens only if the iPhone application is running but in this case recurs every second for every instance of the application running. The process can again be separated into four basic steps. The corresponding diagram is shown in Figure 4.19b.

Step 5

At the time when the user wants to know the current consumption, the eMeter iPhone application requests the data from the web server. It does this by issuing a GET command on the corresponding resource represented by a special URI.

Step 6

The server creates an appropriate request for its database which retrieves the data needed to send a response to the client.

Step 7

The database responses with a tuple. This is transformed by the server into a model object. The JSON representation of this object is then sent back to the iPhone application.

Step 8

The iPhone application parses the JSON encoded message and presents the consumption to the consumer in its user interface.

4.6 Prototype deployment

The UML component diagram depicted in Figure 4.20 serves to model the physical prototype deployment of the system components onto computer nodes.

4.6.1 Smart meter

The prototype deployment of the eMeter system uses the ZMK420 smart meter which is a development prototype provided by Landis+Gyr. It is a follow-up to the ZMK400 [26] which was the first SyM^2 compatible device and as such the first non proprietary meter. SyM^2 is an abbreviation for Synchronous Modular Meter which describes a new generation of electric meters. It was standardized by a consortium of electricity suppliers (EnBW, E.ON, RWE) together with the Physikalisch-Technische Bundesanstalt which is the national metrology institute in Germany. The main features of a SyM^2 device is that it uses a second counter to reliably maintain the correct time-reference even after multiple power failures, uses SML for the communication and is built in a modular way.

Time-reference

Devices conforming to the SyM^2 specification use a second counter which is incremented monotonously. Second counter and current time are bound together after the read-out. This resolves the problems introduced when using a real-time-clock which resets after a power outage. This can make measurements useless when multiple power outages occur between two measurements. The second index needs also not to be corrected for different time zones or daylight saving time.

SML

Advantages of the use of SML for communication is that the data is provided as a one file which is self-explanatory. The SML messages also include signatures which provide

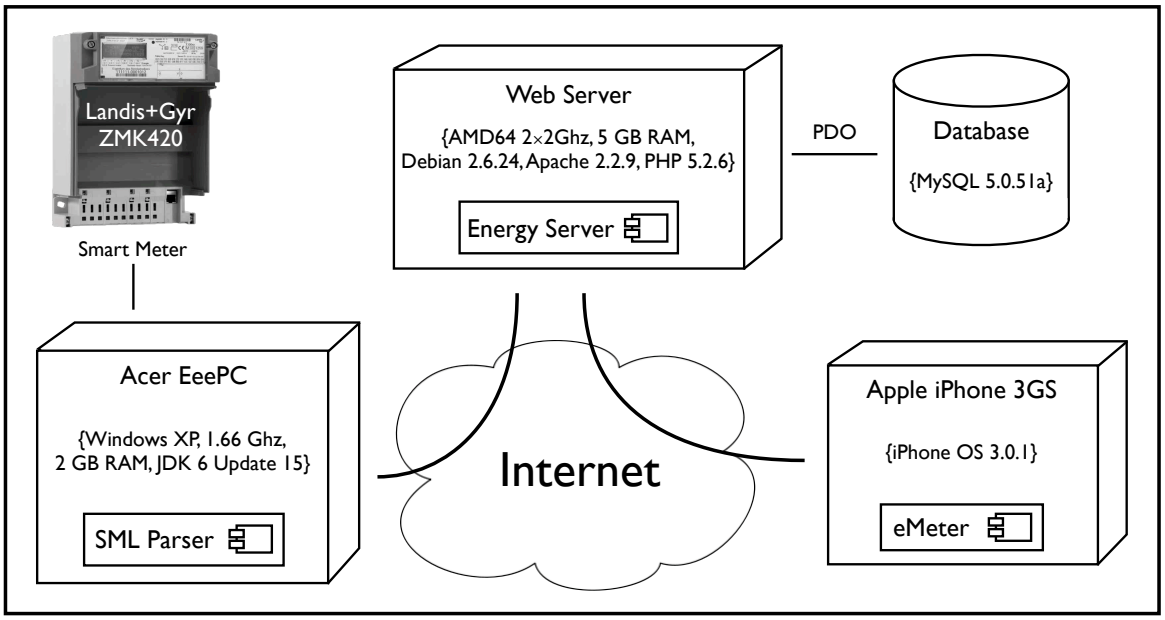


Figure 4.20: UML deployment diagram describing the actual prototype.

the possibility to verify measurement data without dismounting and opening the smart meter and verifying it using the internal log files. Signatures have the additional advantage that the verification is possible even long after the time of measurement.

Modularity

The basic electric meter can be extended with different modules which allow to customize it for a special purpose without the need to built an extra device. Modules are available for instance for pulse forwarding and mobile communication like GSM and GPRS.

4.6.2 Parser

The parser is deployed to an Acer EeePC running Windows XP. The machine has a CPU with 1.66 GHz and 2 GB RAM. JDK 6 Update 15 is installed because the parses is developed in Java. The connection to the smart meter is achieved with a Ethernet cable. Wifi is used for the Internet connection which is needed to send the measurements to the server.

4.6.3 Server

The server application is installed on a standard web server. It is equipped with a 2 GHz AMD64 dual core CPU and 5 GB RAM. The server runs Debian 2.6.24 with Apache 2.2.9 and PHP 5.2.6 installed. The server has a standard MySQL 5.0.51a database. PDO is used as an additional abstraction layer to the MySQL database.

The minimum requirements for the server are Apache 2.2, PHP 5.2.4, an enabled mod-rewrite extension, the MySQL PDO extension and MySQL 5. The server is built using version 0.12 of the Recess framework.

4.6.4 iPhone application

The user interface represented by the eMeter iPhone application is deployed on Apples iPhone 3GS devices running iPhone OS 3.0.1 (build 7A400). It has an ARM Cortex-A8 CPU with 833 MHz slowed down to 600 MHz. The iPhone 3GS is equipped with 256 MB RAM. It can establish its Internet connection over Wifi as well as UMTS to communicate with the server.

5 Conclusion

In this section I will discuss the results and give an outlook on ongoing work.

5.1 Results

Fischer emphasized in 2008 that "failure to meet technological preconditions has stunted the development of useful feedback systems" [12]. This was the starting point for the eMeter system. The goal was to built a system which is both simple to deploy and built only using standard components.

5.1.1 Daily life

The eMeter system integrates feedback on energy consumption into the daily life. By using a mobile phone for the user interface information on power usage is at hand when needed. It allows users to interactively survey their home or monitor their consumption wherever they are and whenever it comes to their mind. When a user intends to leave its home he can check with a quick look if the consumption is in standby range.

The eMeter application is independent of power cables or extra batteries which have to be replaced every so often. It does not require the purchase of a single purpose device which needs space and can soon loose the user's interest.

5.1.2 Lightweight components

The parser was developed using in Java which makes it portable to many different systems. The server is written in PHP and uses a standard MySQL database which allows for the deployment on any off-the-shelf web server. The use of these common components makes the back-end lightweight and reliable.

5.1.3 Standard devices

Instead of requiring the user to buy and install special purpose sensors for every appliance the eMeter system requires only a smart meter and a mobile phone. It can be easily installed once smart meters are available without the need for manipulations of the electric box or new cabling.

By using only standard devices the usage barrier for the proposed system is lowered considerably.

5.1.4 Multilevel feedback

The eMeter system provides multilevel feedback even though it uses only a single sensor. By observing changes of the load curve users cannot only continuously monitor the overall consumption of their homes but also measure single appliances. With the proposed system it is possible to measure devices even if they have no standard AC plug or cannot be unplugged. This would not be possible with inline sensors which have to be plugged between device and outlet.

However, the use of a single sensor has negative effects, too. At the moment it is for example not possible to measure devices which vary in their consumption over time or which cannot be easily turned on or off. To quote Fitzpatrick and Smith again: "Usage by appliance might be desired in principle but not at a cost of a plethora of devices and not necessarily by increasing technical complexity" [13]. In the end the eMeter system enables users to measure most of their devices in a playful way. To me this is an acceptable tradeoff between the detail of the consumption breakdown on one hand and the complexity of the system on the other hand.

5.2 Outlook

In this section I will first describe possible further improvements for each component of the eMeter system. As a second step I will give an outlook on future work.

5.2.1 Parser

At the moment the parser runs on a comparatively powerful netbook which takes space and constantly consumes 13 W. This amounts to 114 kWh per year or approximately 15 €. Therefore it would be better to use an embedded systems instead which is smaller, needs less energy but also has a lower performance. It is planned to use a Gumstix verdex pro in a future prototype. It has a 600 MHz CPU and 64 MB RAM as well as a Wifi module and a LAN port. Its dimensions are only 80 mm × 20 mm × 6.3 mm and it needs only 3 W. This would amount to 26 kWh or approximately 3.50 € per year. In a first step the Gumstix would replace the netbook. Later on, the parser could be deployed directly to an extension module for the smart meter.

Another feature which would be useful for user studies is the remote control over the parser. This would improve the maintainability. In the case of connection problems the parser could cache measurements and sent them in a batch to the server when the Internet access is reestablished.

5.2.2 Server

The server could be extended with graphs and visualizations, comparisons between different users and other community features. This would enable the evaluation of comparative feedback. Another possibility would be to integrate the eMeter system into social networks as a plugin.

Device recognition

At the moment devices are managed in the iPhone application. This could be shifted to the server to built up a data base of different devices and their load curve characteristics. Once the data basis is big enough it could be used to tell device categories apart. On the basis of this knowledge the eMeter system could provide information on the performance of a device compared to other devices in the same category.

5.2.3 Visualization

The iPhone application could recommend a device category after measuring an appliance which fits the load curve change best. As a result personalized energy saving tips could be given based on the device category. Recommendations for the purchase of a more efficient appliance which would replace an old one which wastes energy and money would become possible.

Further visualizations could be developed for TVs using AppliCast or picture frames using RSS. Other mobile phone platforms like Android could also be supported with native clients.

5.2.4 Future work

The work on the eMeter system is continued as an ETH ecoworks[1] project. Ecoworks projects have the goal to increase the energy efficiency of the ETH.

One of the next steps is the deployment of the eMeter system within a user study. The eMeter system will be installed in the iHomeLab which is a research laboratory for intelligent living and building automation of the Lucerne School of Engineering and Architecture. This allows to evaluate real life usage information of the iPhone application and test the overall reliability of the system. Interesting aspects to test are for instance different approaches to device measurement or how often feedback has to be given to be effective.

[1] `http://www.ecoworks.ch/projects/detail/248`

Bibliography

[1] W. Abrahamse, L. Steg, C. Vlek, and T. Rothengatter. A review of intervention studies aimed at household energy conservation. *Journal of Environmental Psychology*, 25(3): 273–291, September 2005.

[2] Apple Inc. iPhone Developer Center. Website. `http://developer.apple.com`.

[3] Apple Inc. iPhone OS Overview. Website, June 2009. `http://developer.apple.com/iphone/library/referencelibrary/GettingStarted/URL_iPhone_OS_Overview/`.

[4] Blue Line Innovations Inc. Blue Line PowerCost Monitor. Website. `http://www.bluelineinnovations.com`.

[5] P. B. Crabb. Effective control of energy-depleting behavior. *American Psychologist*, 47 (6):815–816, June 1992.

[6] S. Darby. The effectiveness of feedback on energy consumption. *A Review for Defra of the Literature on Metering, Billing and Direct Displays*, April 2006.

[7] DIY Kyoto. Wattson. Website. `http://www.diykyoto.com`.

[8] Electronic Educational Devices. Watts Up. Website. `http://www.wattsupmeters.com`.

[9] Energy Inc. The Energy Detective. Website. `http://www.theenergydetective.com`.

[10] European Parliament and Council. Directive on energy end-use efficiency and energy services, article 13, para. 2 and para. 3(a,b). Directive 2006/32/EC, April 2006. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2006:114:0064:0064:EN:PDF`.

[11] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. `http://tools.ietf.org/html/rfc2616#section-10`.

[12] C. Fischer. Feedback on household electricity consumption: a tool for saving energy? *Energy Efficiency*, 1(1):79–104, February 2008.

[13] G. Fitzpatrick and G. Smith. Technology-enabled feedback on domestic energy consumption: articulating a set of design concerns. *IEEE Pervasive Computing*, 8(1): 37–44, 2009.

[14] G. Gardner and P. Stern. *Environmental Problems and Human Behavior*. Pearson Custom Publishing, Boston, 2nd edition, 2002.

[15] E. S. Geller. Evaluating energy conservation programs: Is verbal report enough? *The Journal of Consumer Research*, 8(3):331–335, December 1981.

[16] German Federal Ministry of Justice. Energiewirtschaftsgesetz, article 21b, para. 3a. Juris, July 2005. `http://bundesrecht.juris.de/enwg_2005/__21b.html`.

[17] G. W. Hart. Residential energy monitoring and computerized surveillance via utility power flows. *IEEE Technology and Society Magazine*, pages 12–16, June 1989.

[18] S. C. Hayes and J. D. Cone. Reducing residential electrical energy use: Payments, information, and feedback. *Journal of Applied Behavior Analysis*, 10(3):425–435, Fall 1977.

[19] E. Hinterbichler. Designing a better energy consumption indicator interface for the home. *Master Thesis at the University of Illinois at Urbana-Champaign*, 2008.

[20] J. M. Hunt, J. F. Holmes, R. S. Carr, and J. DaIzell. Electrical energy monitoring and control system for the home. *IEEE Transactions on Consumer Electronics*, 32(3): 578–583, August 1986.

[21] IEEE Global History Network. Pearl street station. Website, September 2009. `http://www.ieeeghn.org/wiki/index.php?title=Pearl_Street_Station&oldid=32794`.

[22] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler. Design and implementation of a high-fidelity ac metering network. International Conference on Information Processing in Sensor Networks, 2009.

[23] K. Jordan. Recess PHP Framework. Website. `http://www.recessframework.org`.

[24] Y. Kim, T. Schmid, Z. M. Charbiwala, and M. B. Srivastava. Viridiscope: Design and implementation of a fine grained power monitoring system for homes. UbiComp, 2009.

[25] R. Kohlenberg, T. Phillips, and W. Proctor. A behavioral analysis of peaking in residential electrical-energy consumers. *Journal of Applied Behavior Analysis*, 9(1):13–18, Spring 1976.

[26] Landis+Gyr. ZMK400. Website. `http://www.landisgyr.eu/de/pub/produkte_und_loesungen.cfm?eventProducts=products.ProductDetails&ID=57&catID=9`.

[27] J. Lifton, M. Feldmeier, Y. Ono, C. Lewis, and J. A. Paradiso. A platform for ubiquitous sensor deployment in occupational and domestic environments. *International Conference on Information Processing in Sensor Networks*, pages 119–127, April 2007.

[28] L. McClelland and S. W. Cook. Energy conservation effects of continuous in-home feedback in all-electric homes. *Journal of Environmental Systems*, 9(2):169–173, 1979.

[29] A. Meyel. Low income households and energy conservation: Institutional, behavioural and housing barriers to the adoption of energy conservation measures. *Built Environment Research Group, Polytechnic of Central London*, 1987.

[30] P3 International. Kill A Watt. Website. `http://www.p3international.com`.

[31] D. S. Parker, D. Hoak, and J. Cummings. Pilot evaluation of energy savings from residential energy demand feedback devices. Final Report by the Florida Solar Energy Center to the U.S. Department of Energy. FSEC-CR-1742-08., January 2008.

[32] D. Petersen, J. Steele, and J. Wilkerson. Wattbot: A residential electricity monitoring and feedback system. Conference on Human Factors in Computing Systems, 2009.

[33] J. E. Petersen, V. Shunturov, K. Janda, G. Platt, and K. Weinberger. Dormitory residents reduce electricity consumption when exposed to real-time visual feedback and incentives. *International Journal of Sustainability in Higher Education*, 8(1):16–33, 2007.

[34] Population Division, U.S. Census Bureau. Population estimates program, historical national population estimates. Website, April 2000. `http://www.census.gov/popest/archives/1990s/popclockest.txt`.

[35] Population Division, U.S. Census Bureau. Annual estimates of the resident population for the united states. Website, December 2008. `http://www.census.gov/popest/states/NST-ann-est.html`.

[36] K. Roth and J. Brodrick. Home energy displays. *ASHRAE Journal*, pages 136–138, July 2008.

[37] C. Seligman and J. M. Darley. Feedback as a means of decreasing residential energy consumption. *Journal of Applied Psychology*, 62(4):363–368, August 1976.

[38] R. J. Sexton, N. B. Johnson, and A. Konakayama. Consumer response to continuous-display electricity-use monitors in a time-of-use pricing experiment. *The Journal of Consumer Research*, 14(1):55–62, June 1987.

[39] Sun Java Bug Database. ForceTimeHighResolution switch doesn't operate as intended. Website, June 2006. `http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6435126`.

[40] SyM^2 consortium. Smart message language, version 1.02. Website, January 2008. `http://www.t-l-z.org/docs/SML_080711_102_eng.pdf`.

[41] SyM^2 consortium. SML protocol for wide-area transmission. Website, November 2008. `http://www.t-l-z.org/eng/sml_eng.html`.

[42] T. Ueno, F. Sano, O. Saeki, and K. Tsuji. Effectiveness of an energy-consumption information system on energy savings in residential houses based on monitored data. *Applied Energy*, 83(2):166–183, February 2006.

[43] United States Department of State. Second arab oil embargo, 1973-1974. Website, September 2009. `http://www.state.gov/r/pa/ho/time/dr/96057.htm`.

[44] United States Energy Information Administration. Annual energy review. Website, 2008. `http://www.eia.doe.gov/aer/`.

[45] Visible Energy Inc. UFO Powerstrip. Website. `http://visiblenergy.com`.

[46] Wikipedia, the free encyclopedia. Bluetooth. Website, September 2009. `http://en.wikipedia.org/w/index.php?title=Bluetooth&oldid=312478543`.

[47] Wikipedia, the free encyclopedia. JSON. Website, September 2009. `http://en.wikipedia.org/w/index.php?title=JSON&oldid=312412842`.

[48] Wikipedia, the free encyclopedia. Mach. Website, September 2009. `http://en.wikipedia.org/w/index.php?title=Mach_(kernel)&oldid=307518636`.

[49] Wikipedia, the free encyclopedia. Network address translation. Website, September 2009. http://en.wikipedia.org/wiki/Network_address_translation.

[50] Wikipedia, the free encyclopedia. PHP data objects (PDO). Website, September 2009. http://en.wikipedia.org/w/index.php?title=PHP&oldid=312488445.

[51] Wikipedia, the free encyclopedia. Power line communication. Website, September 2009. http://en.wikipedia.org/w/index.php?title=Power_line_communication&oldid=313585920.

[52] Wikipedia, the free encyclopedia. Representational state transfer (REST), September 2009. URL http://en.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=312399633.

[53] Wikipedia, the free encyclopedia. Smart meter. Website, September 2009. http://en.wikipedia.org/w/index.php?title=Smart_meter&oldid=311596925.

[54] Wikipedia, the free encyclopedia. Unified modeling language (UML). Website, September 2009. http://en.wikipedia.org/w/index.php?title=Unified_Modeling_Language&oldid=311781476.

[55] G. Wood and M. Newborough. Dynamic energy-consumption indicators for domestic appliances: environment, behaviour and design. *Energy and Buildings*, 35(8):821–841, September 2003.

[56] T.-J. Yun. Investigating the impact of a minimalist in-home energy consumption display. Conference on Human Factors in Computing Systems, 2009.