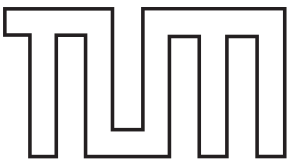

Technische Universität München

Distributed Multimodal Information Processing Group

Prof. Dr. Matthias Kranz

Diplomarbeit

System zur Unterstützung effizienterer Seminarraumnutzung

Author:	Tobias Knothe
Matriculation Number:	[redacted]
Address:	[redacted]
	[redacted]
Advisor:	Andreas Möller
Begin:	01.04.2011
End:	30.09.2011

Abstract

In large organizations like offices, universities or generally shared real estate a common administrative problem is that of physical resource scheduling. Often resources like meeting rooms are too plentiful to be managed by one central authority and so the process of acquiring a room is inconsistent, complicated and awkward. This thesis attempts to provide a scalable solution using the ubiquitous computing paradigm to handle administrative tasks related to resource management in a consistent and automated fashion. Innovative features using state-of-the-art technologies provide security, ease of use and added value to both end-users and resource administrators. The practical part of this thesis is a prototype implementation of the system encompassing a backend server infrastructure, portable terminal clients bound to the room resources and embedded hardware device directly interacting physically with the environment.

Contents

Chapter 1

Introduction

1.1 Motivation

With the rapid expansion of faculties and the quickly increasing number of students studying at the Technische Universität München (TUM) around 2010, a major problem for the university's administration became that of scheduling rooms and lecture halls. The main campus site could not expand any further and plans for moving to the new site would take years. Therefore it quickly became apparent that the existing, decentrally managed infrastructure needed to be used more efficiently. First thoughts on developing an assistance system which employs an ubiquitous computing model were formed by the Distributed Multimodal Information Processing Group at the TUM. This thesis attempts to provide a conceptualization and specification of such a system in this written part, along with a description of the first working implementation, which was implemented in the second, practical part. This implementation, called the TUManage system, is currently being deployed in a productive test-phase to manage rooms in one building of TUM's Munich inner city campus.

1.2 Outline

In the next chapter a brief look is taken at related work upon which this thesis is built. Chapter 3 describes the user studies done to profile the system's target audience and secify user requirements. Chapter 4 deals with the main part of the actual system, the front-end consoles with which the user directly interacts. A brief look at usability issues and design is taken before the platform implementation is explored, whereafter a UI walk-through is taken. In the next chapter, chapter 5, the design and implementation of the supporting backend server infrastructure is examined. Chapter 6 deals with how the system interacts with the user's physical environment. Finally, in chapter 7 a conclusion and outlook is presented.

Chapter 2

Related work

TUManage's usability and architecture design leans on current research in the field of ubiquitous and pervasive computing. An introduction to ubiquitous computing is provided in the section below.

2.1 Ubiquitous computing

2.1.1 Terminology

The term "ubiquitous computing" is a human-computer interaction model used to describe the third, present era in modern computing [20]. The first era was defined by the idea of the mainframe server where computing time was made available across multiple "dumb" terminals. This large computer was often owned by an organization and shared by many people at the same time. Second, came the era of the Personal Computer (PC) - one computer owned and used by primarily one person. The "Third Paradigm", as Alan Kay of Apple calls it [29], is termed ubiquitous computing, or the age of "calm technology" [30] (when technology recedes into the background of our lives).

2.1.2 Usability

As opposed to the PC, one person now interacts with a multitude of different computers and systems, seamlessly integrated into everyday objects and our everyday life. These small networked portable devices can take on the form of smartphones, household appliances with embedded computers (e.g. television sets) or other forms of embedded devices. The term Internet of Things is used in conjunction with these devices because of their network connectivity and pervasive presence [22]. The fact that many of these highly integrated devices can be connected together logically to form a complex system present in many parts of our everyday environment opens up new opportunities for "machines that fit the human environment instead of forcing humans to enter theirs" [31]. For the user, interaction with

the systems becomes more and more transparent, the user has no knowledge of the presence or complexity of system components running in the background to complete the task at hand. He can engage with the system in an unobtrusive and natural way while being in contact with many parts of the system at the same time.

The devices should recede into the background of the user's perception, with all complexity being abstracted and hidden from the user. Users typically engage with ubiquitous systems spontaneously and not want to invest time to learn operation. As the systems are embedded into everyday objects most users are in a different mindset when interacting with them as when they were, say, using a PC, where an average user is used to adapting to the system. Here, the system needs to adapt to the user (calm technology [30]), and stay out of the user's way. Calmness, as coined in [30], is a new challenge for system usability designers. Systems are no longer used by a small set of experts, they are used by everyday users in everyday situations. PCs focus more on excitement of interaction. Calm systems should rather be less noticed.

2.1.3 Interactivity

Input data is gathered from the user and the environment by means of numerous embedded and mostly unnoticed sensors. This makes the system context-aware, it can change its behavior depending on numerous factors like e.g. the environment or location it is being used in, the type of user using it or the proximity of other devices part of the system. Context awareness can span across multiple levels and components of the system. Output can be actions on the environment using actuators (e.g. opening a door). Information can be presented to a user in a form best matching his conceptual model of the system (e.g. turning a door handle green if the door is unlocked, red if it is locked).

2.1.4 Future

With embedded devices becoming more and more powerful and affordable, ubiquitous computing can ever easily be integrated into larger-scale everyday systems. The explosion in smartphone sales in the recent years alone has brought powerful hand held devices with flexible computing platforms to the consumer market. Also, on the sensor/actuator side an ever increasing number of intelligent embedded devices with excellent networking capabilities have become available. Wireless Sensor Network (WSN)s and pervasive systems are becoming a trend and have been recognized by numerous commercial platform developers, as can be seen with e.g. the many ZigBee implementations or Google's Android@Home project [18].

Chapter 3

User study

Before designing the new room management system a user study was conducted, consisting of both user interviews held in person and an online survey with anonymous participants. From these investigations a set of key requirements were distilled as well as a profile about the system's target audience established.

3.1 User interviews

The first user interviews were held in person in order to get a selected but very detailed response. The members of both the TUM staff plus administration and the students were very clear to highlight the currently problematic room situation at TUM's inner-city campus. For one, rooms are very limited and there are little common areas available for collaboration, while on the other hand smaller lecture halls and seminar rooms remain unused for long periods of time. Different rooms are managed by different administrative entities within TUM so even if a possibly free room would be available, users do not know if they may use it or whom to ask. Often rooms are locked, with only a very small subset of staff having a key. Other potential users would need to fetch the key at some hard to find or unknown office. Room administrators expressed concern about moving room ownership to a central administration as they fear they themselves would be obstructed from using the room efficiently. This could take on the form of double-bookings or the room being inaccessible for spontaneous ad-hoc meetings due to the high overhead of booking the room and obtaining the key from the central room administration.

From these user interviews it was concluded that a homogeneous and simple means of managing rooms needed to be introduced. The same procedures for using rooms should be valid across campus in order to avoid confusing users. The problem of high overhead to obtain a room has to be addressed. Reserving a room should be a quick and easy procedure and instant feedback must be given, instead of the usual one or two days waiting time for the room administrator to reply to a user's email inquiry.

Some students interviewed also expressed their dissatisfaction with the paper room sched-

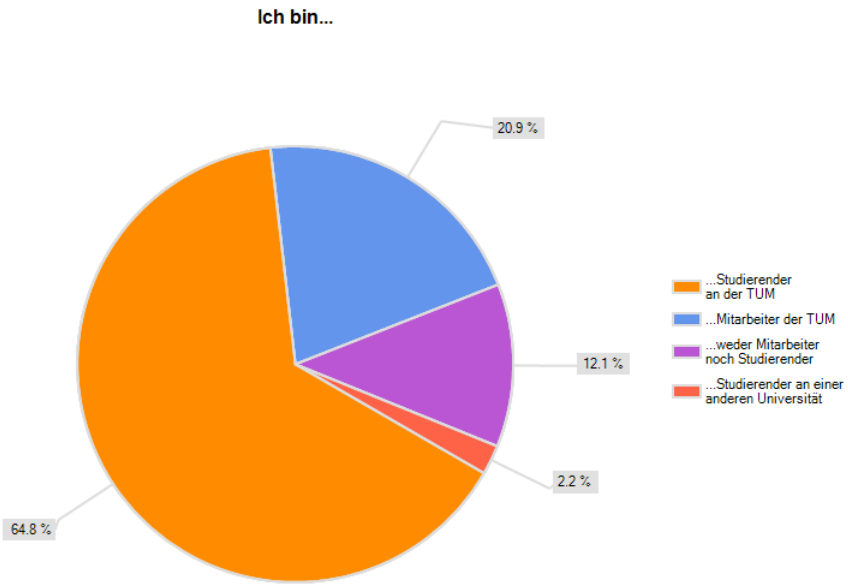

Figure 3.1: Participant role at the TUM

ules pasted next to the lecture hall doors. In theory they should show the event timetable for the room, but the static weekly schedule does change sometimes, making the paper sign confusing. The paper sign is also said to be printed too small.

3.2 Online survey

Next, an online survey was conducted online using a popular 3rd-party web-based survey tool [8]. The goal of this survey was to get a broad overview of possible users at TUM and highlight possibilities for assisting with everyday tasks in the campus ecosystem. A profile of the participants' use of mobile Internet, smartphone use and use of web platform was made in order to best cater for their needs while designing the room management system. Questions with regards to room use at the TUM campus were also asked as well as some questions profiling the willingness to adapt new technologies for everyday tasks. Here is a selection of interesting results from the survey.

A total of 93 participants completed the survey. About 65% were TUM students and 21% TUM staff (fig. 3.1).

Over 75% of participants has access to the Internet using their mobile phone (fig. 3.2).

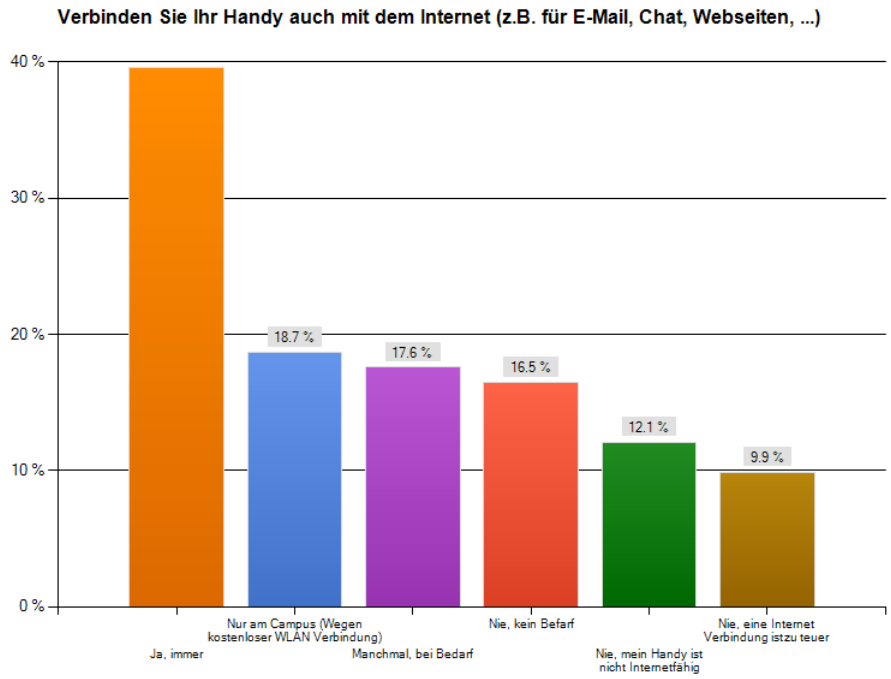

Figure 3.2: Mobile phone Internet usage

Users of Internet-enabled mobile phones use their mobile's connectivity mostly for email, followed by Web 2.0 networks, news and navigation (fig. 3.3). Many users would like to use it for navigation within buildings and for mobile payment, but only within the TUM campus. A surprisingly high number of participants (35%) would like to use thier mobile phone to control everyday items like their TV (or coffee machine). While a large part of the target audience has an Internet-enabled mobile phone this percentage does not cover the entire target audience.

When asked about online service usage at the TUM the most used service was the canteen's online lunch menu, followed by the use of the TUM Internet portal used by students and staff for organizational tasks like registering for courses, managing course notes and checking one's course schedule (fig. 3.4). Room reservation services scored very low, not surprising as there was no official service provided at the time the survey was held.

The response for the question about which new online services at TUM would be desired was mixed (fig. 3.5). It does seem, however, that a mobile room finder and navigator are the most desired services. Interestingly, the proposal of an online room reservation system has the highest neutral response. Users do not explicitly wish for the system but seem open for its use. (It should be noted that at this part of the survey there was no mention yet of the possibility of using rooms for personal collaboration or studying by students.)

The last set of questions were about room use at the TUM. There was moderate agreement

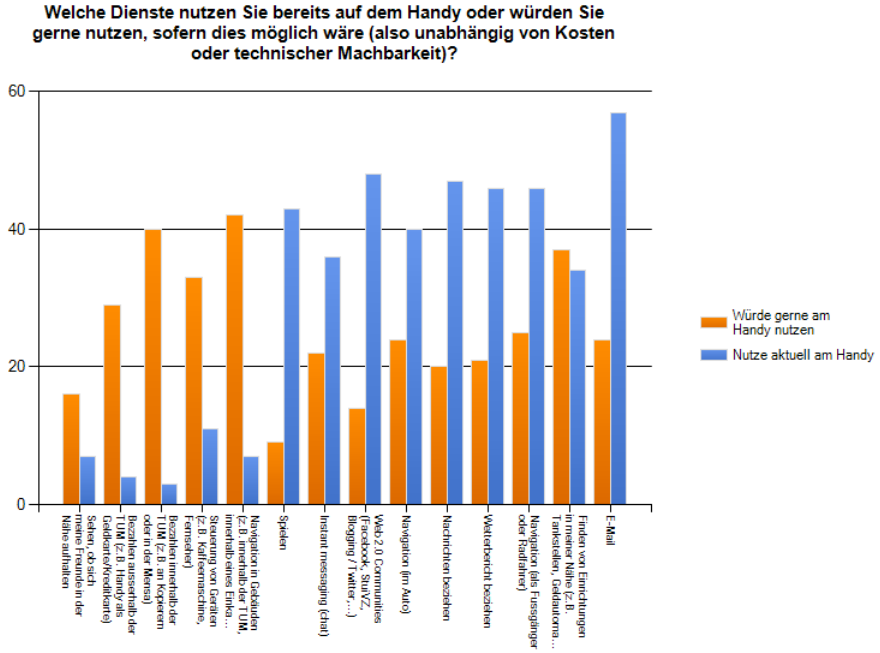

Figure 3.3: Mobile phone services usage

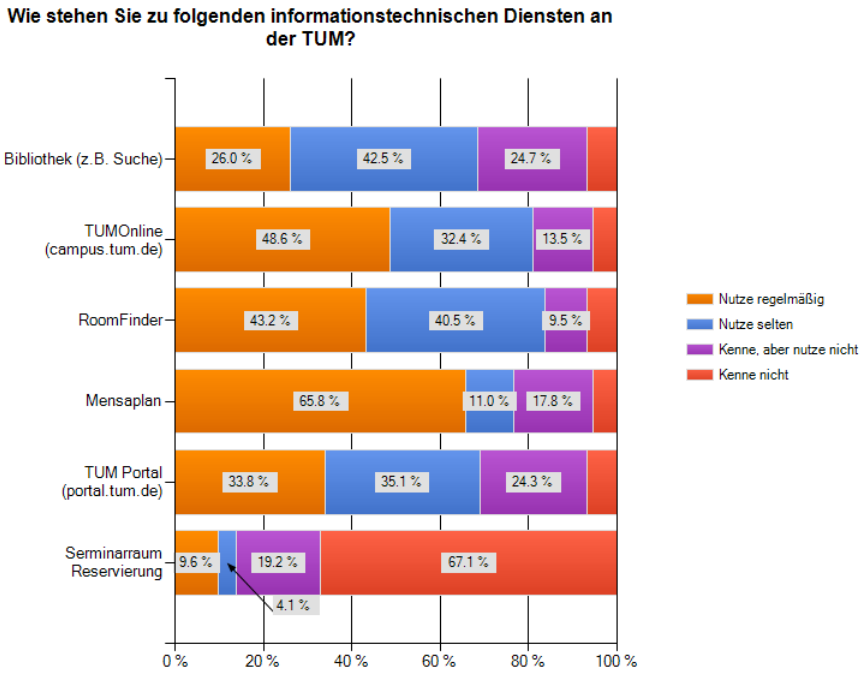

Figure 3.4: Current TUM online services usage

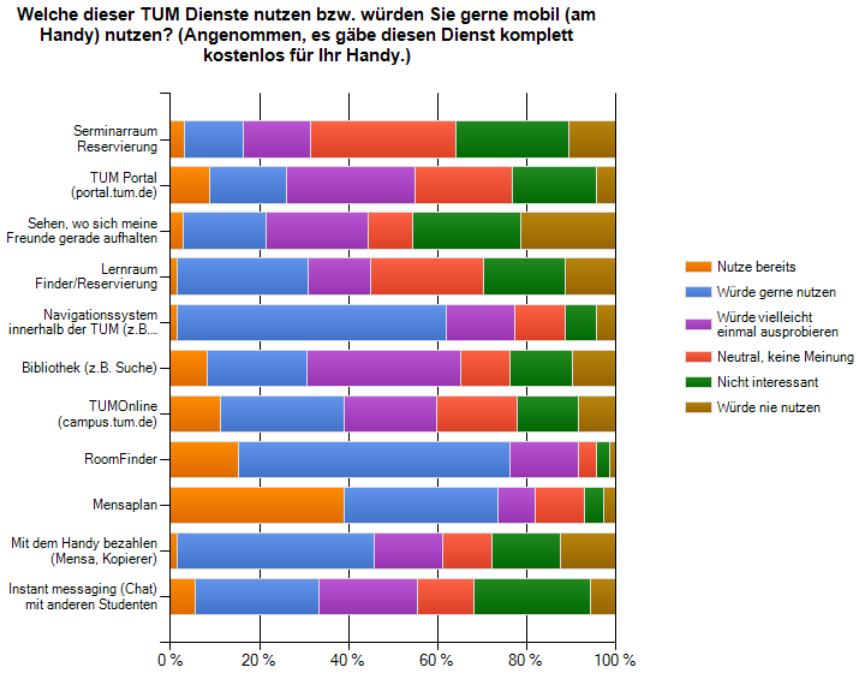

Figure 3.5: Desired TUM online services usage

that rooms are easy to find at TUM and strong agreement that the TUM RoomFinder web service is a valuable tool for finding unknown rooms (fig. 3.6). Moderately strong disagreement was observed for the statement that there are enough rooms available for ad-hoc collaboration, meetings or studying. This confirms the statements about the problematic rooms situation encountered during the personal interviews.

Asked in more detail about the reason for this dissatisfaction the number one answer was that rooms are not free when needed (fig. 3.7). This is interesting because there are, according to the campus administration, many free rooms, but users do not know about them. The next prevalent answers were, apart from "the rooms are too uncomfortable", that "i don't want to be in anyone's way" and "i didn't know i was allowed to use the room". Also interesting are the relatively common answers "the rooms are often locked" and "i wish to work without being disturbed". The answers to the survey highlight some important issues the new room management system must address. Free rooms should be advertised to users and they must be sure that it is allowed for them to use the room. While using a room, users wish to not be disturbed and be left assured that they can use the room without interruption for a certain amount of time. It is therefore proposed to implement the policy that rooms have to always be reserved, even if they are to be used immediately. Once a user has a reservation in his name he can firstly be assured that he may use the room. Secondly he will know that he can use the room without interruption, knowing at what time his reservation ends and the next reservation begins.

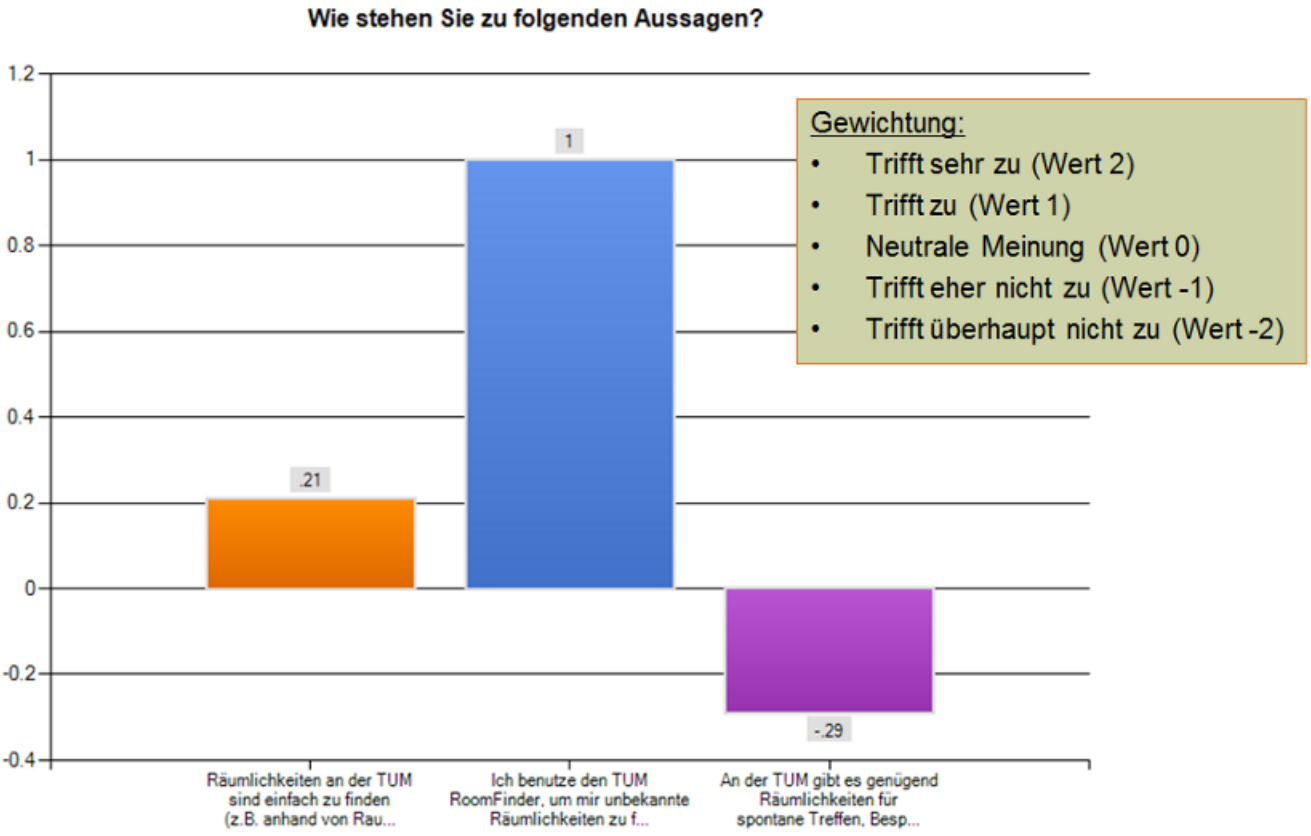

Figure 3.6: Satisfaction levels about room situation

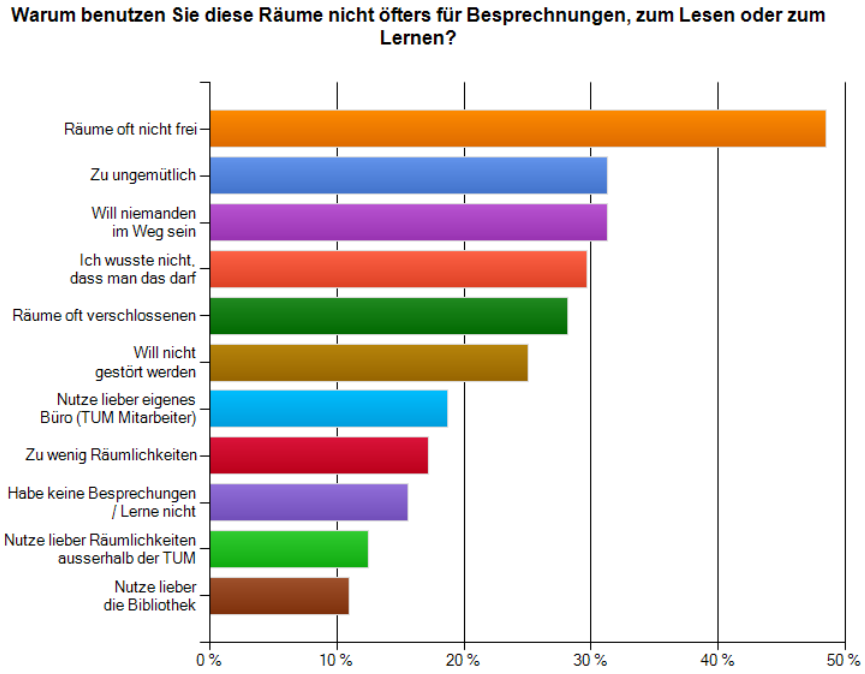

Figure 3.7: Reasons for not using rooms

3.2.1 Outcome

Some key requirements for the new room management system could be defined with the help of the interviews and the survey. The new system should,

- Assist with managing the numerous rooms at TUM belonging to different administrative domains.
- Enable rooms to be used more efficiently by offering them to any user, from any administrative domain, when the room would otherwise remain unused.
- Cut the time and effort required by users to find and use a suitable room to fit their needs.
- Keep a record of the identity of users reserving a room, in case of abuse or theft.
- Implement an automatic access control mechanism allowing users who have reserved the room to enter it immediately, replacing the current physical keys which are cumbersome to administer.
- Have a consistent look-and-feel across the entire campus and require no training for its users.
- Allow users to quickly and reliably verify a room's schedule or book it. No stale data

or high latency can be tolerated.

- Accidental double-bookings due to simultaneous booking requests need to be avoided, while still keeping reservation overhead at an absolute minimum.

Following these key requirements a room management system was designed and its name, TUManage, coined. It consists of three main parts, namely terminals called WallClients mounted next to rooms' doors, replacing the traditional door signs found at TUM. These will be the main point of user interaction and provide a familiar and homogenenous user experience at all room terminals throughout campus. The second part are small embedded devices called RoomControl which interact with the room environment, in this case the room's door and lock. TUManage can therefore unlock doors automatically and sense the state of doors and locks, automating the process of fetching keys from room administrators. The third part is a server infrastructure connecting all the WallClients and RoomControls together. Here the room management is handled centrally, with the opportunity to integrate with existing systems like the TUM's university management system TUMonline.

Chapter 4

WallClient displays

One of the components of the TUManage system is the smart door sign. This chapter deals with its design and implementation. A prototype implementation was installed at one of TUM's lecture rooms for demonstration and evaluation purposes.

4.1 Overview

Direct user-interaction with the TUManage system is usually primarily done via a dynamic door sign device called the WallClient. It is placed next to each of the room's main doors, replacing the traditional door signs displaying the room's name and number. Interactivity with the sign is achieved using a touch-sensitive display, allowing the users to navigate around the custom Graphical User Interface (GUI) (WallClientUI) in an intuitive and familiar way. The second means of interactivity is provided by the system's backend services which can respond to environmental (e.g. the door is unlocked) or logic events (e.g. a room reservation has taken place).

4.2 Hardware platform

4.2.1 Hardware requirements

Requirements for the hardware hosting the WallClientUI are simple. The device should be mounted in an accessible location next to the rooms' doors. It should be unobtrusive and easy to use, therefore a touch-screen display is required. The display should be well-readable even when it is mounted on a wall while users stand in front of it. The device should be easy to install, some installation locations have no means of routing Local Area Network (LAN) cabling to the device. Power cabling, is often too thick to install easily or this is not possible due to safety regulations, even though power is usually available nearby (e.g. from plug sockets, light switches or light fittings). Therefore low-voltage

power supply cabling would be advantageous as it can be much thinner and is not subject to strict building safety regulations.

4.2.2 Android platform

The Android platform was chosen as best suited platform for the WallClient devices. With the availability of Android tablet devices on the mass-market hardware can be obtained quite cheaply. Newer Android tablets offer impressive technical specifications like 3D graphics accelerators and high-performance processors in a small and simple form-factor. The Android operating system is well-suited to be adapted for WallClient devices as it is open-source, allowing modifications and customizations to be performed easily. Even without modifications the framework offers many useful features (like the intent framework, Extensible Markup Language (XML)-based layout, package deployment) and a powerful Integrated Development Environment (IDE) which make the platform very well suited for rapid prototyping.

4.2.3 Connectivity

Because the WallClients are explicitly mounted next to rooms' doors it could be a challenge to supply the device with power and network connectivity. The Android devices are well suited for the task for two reasons. Firstly, their network access is handled via a built in Wireless LAN (WLAN) chipset, eliminating all need for running dedicated network cabling. The devices connect to the building WLAN infrastructure which is omnipresent on-site. Secondly, the devices do not require 220V mains power. They are supplied with 5V-19V DC, depending on make and model. Power feeds can therefore be safe low-voltage cables which are easy to run both practically and legally. The 220V power supply can be mounted out of reach and sight at a convenient location.

4.2.4 Operating-system lockdown

In order to prevent malicious users from manipulating the device software or abusing the system, measures were taken to lock the user into the WallClient software. Access to the underlying Android operating system is not possible as all elements of the Android user-interface leading away from the WallClient application are either removed or disabled. Also, automatic sleep or screen lock are disabled programmatically so the device is kept awake and the display on at all times. Should the WallClient application for some reason be terminated, a service responds to this condition and will restart it.

4.2.5 Software deployment

Once many WallClient devices are in operation a means of automatic deployment and device monitoring is essential for uninterrupted and efficient operation. While the WallClient application is designed to be low-maintenance, with large parts of the system's complexity shifted to the core backend services and using a code-download architectural style for often changing parts, it is inevitable that maintenance needs to be performed and administrator intervention is essential. Future versions of the TUManage system could employ a device management service to report device status to a maintenance service or automatically deploy new/updated software packages over the network. The management service should run independently of the WallClient application in order for updates to the WallClient software not to conflict with the management software. Access to system settings like network or display settings would be of advantage. In our prototype system deployment was carried out manually using the device's web browser which was navigated to the WallClient application package hosted at an accessible URL. Settings like the device ID were entered manually.

4.2.6 Anti-theft measures

Even though the relative cost of an Android tablet device is low compared to other solutions it can still be an attractive target for petty thieves. Therefore a tamper-proof wall-mount casing was chosen, making it difficult to remove the device from the wall without generating much attention. Device movement is detected using its built-in acceleration and magnetic compass sensors and it will emit an alarm sound if it is moved. Additionally it will send an alert message to the backend core service with its location for as long as the internal battery lasts.

4.2.7 Building safety requirements

A safety-support feature added to the WallClient software is that it will display an emergency escape route symbol when the main power is cut, until the power is restored or the internal battery runs out. The direction to the nearest escape route is shown with the display backlight on at the highest brightness in order to assist users in the case of an emergency or power failure. (The corresponding wall-client view is shown in figure 4.1.)

4.3 Design principles

The design of the WallClient UI has its primary focus on simplicity, clarity and flexibility. The majority of TUManage users will receive little or no training on the operation of

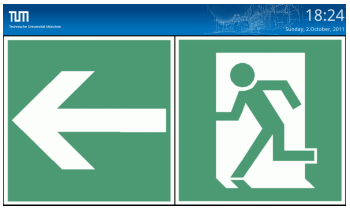

Figure 4.1: Emergency view (shown in the event of a power failure)

the WallClients. The devices should, as any good ubiquitous computing system (see 2.1), recede into the background of the user's perception and "just magically work", with all complexity being abstracted and hidden from the user. Most users will use the system spontaneously and not want to invest time to learn its operation. As it is embedded into an everyday object most users are in a different mindset when interacting with it, as when they were, say, using a PC, where an average user is used to adapting to the system. Here, the system needs to adapt to the user (calm technology [30]), and stay out of the user's way.

This section highlights some of the core principles taken into account while designing the WallClient UI.

4.3.1 Simplicity, Clarity, Flexibility

These three core values were borrowed from a related ubiquitous system design [21]. Simplicity means that the system should not be over-engineered or over-specialized. clarity is needed to convey a unfamiliar idea (i.e. the device's operation). Flexibility is important for further development and adaptation of the system, e.g. when enhancements are to be performed, new requirements arise or the environment changes.

4.3.2 Visibility

Merely by looking at the device the user should be able to tell the state of the system and know his alternatives for action. The navigation bar on the right is kept deliberately simple and maps the user's conceptual model of the system (i.e. his impression and expectations). At all times can the user see all his options, namely to see information about the room (what is its number? is it in use? who is using it or when is it used next?), see the room's schedule and to make a reservation. A good placement and visibility of these controls is vital. While performing more complex operations, like reserving a room, the user is also presented visually with a complete overview of the process (see 4.4.7). As he enters the required information as asked by the system he sees it placed within the context of the task at hand, via the overview control displaying the completed and still pending steps for performing the reservation.

4.3.3 Conceptual Model

The design should provide a good conceptual model for the user, with consistency in the presentation of operations and results and a coherent, consistent system image [23]. The user knows, from the fact that the device is hanging next to a door where a door sign (and in TUM the paper weekly event schedule) usually reside, that the device is used for displaying information about the room and possibly its events. The user mentally simulates the operation of the device before even touching it. The UI should adapt to this model and provide further clues as to the operation by offering affordances, constraints and mappings [23].

Affordances

Affordance refers to the perceived and actual properties of the thing or device, primarily the actual properties which determine precisely how the device is to be used. Affordances provide strong clues to the operation of things. An example of an affordance would be the touch-screen display. The user knows from previous experience with touch devices and the virtual buttons displayed on the screen (and the lack of any other controls) that interactivity with the screen is via touch.

Constraints

Constraints would be the physical space on the screen used by the buttons. The prominent design of the buttons in respect to the other UI elements suggests that these buttons are for touching.

Mappings

Mappings are very important with the touch-screen UI as the only ways to convey the function of a virtual button is using text and position. Therefore care was taken to keep the navigation buttons on the right present at all times when navigating through the UI. The user knows his current location within the UI because the corresponding button is highlighted in the same color as the UI view's background (see fig. 4.3). If at any time the user wishes to navigate somewhere else the required buttons are where he would expect them.

4.3.4 Feedback

The user should receive continuous and full feedback about the results of his actions. Wide use of a so-called "spinning wheel" throbber (see fig. 4.2) is made due to the indicator's

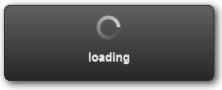

Figure 4.2: Throbber animation

prominence in typical Web 2.0 applications our target audience is familiar with. Sometimes (e.g. during high network and/or server load) it is possible for the system to perform slower than desired and actions take more than the upper bound of 3 seconds to complete. The use of the animated loading indicator conveys the impression that the task has been understood by the system and it is busy processing it, while no further interaction is needed.

4.3.5 Metaphoric Design

The cognitive approach chosen, according to [10], is the metaphoric approach. Using metaphors can be an effective way to communicate an abstract concept or procedure to users, as long as the metaphor is used accurately. When users are confronted with the metaphor of, for example, the switch-like graphic together with the lock symbol of the privacy-control in the reservation view, they use their past experience with switch-like objects in the real world to instantly know that this is a control which can have two togglable states. Other examples of metaphors used in the UI are the schedule control which looks like a timetable and the event progress control which looks like a progress bar filling up.

4.4 User interface implementation

Implementation of the UI was done using two frameworks and architectural styles, the native Android domain and the code-on-demand domain. The next section will briefly deal with the two domains, whereafter the UI is presented in the following sections, view by view. All views, irrespective of their underlying domain, have a few important key points in common, the TUM corporate design [9] and the navigation control, the NavBar. The corporate design is important not only for branding the system in the context of the customer (in this case the TUM), but also for recognition and assurance for the user. A new user might notice the TUManage WallClient with the familiar branding of the building administration and immediately be assured that this is an official and reliable service with support available in case of difficulties. The branding represents the same entity the user would contact via other means for the same outcome, namely contacting the room administration.

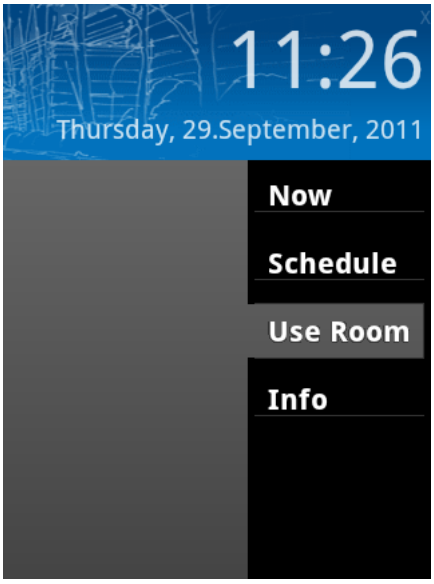

Figure 4.3: NavBar navigation metaphor

4.4.1 Navigation

Special care has been taken to present the user with a clear and consistent base navigation metaphor. It is present on the screen at all times, in all views, in the same look, feel and position. It is positioned on the right of the screen and is touch-responsive. The right side of the screen was chosen because, as a typical user uses his right hand to touch a control, he will thereby not obstruct view of the screen with his right hand and arm. A user touches the corresponding button to switch the main display area on the left of the NavBar to the desired view. While navigating the UI the NavBar adapts its state to show the current context of the UI. This is achieved by highlighting the corresponding button with the background color of the view being used on the left, creating the impression that the button is part of the view in question (even though programmatically it is not). The other buttons seem to hover on the right side of the screen, ready for the user to select them to jump to a different view/task. A prototype version of the NavBar can be seen in figure 4.3, depicted here without tell-tale icons and style, which was added later. The NavBar is implemented in the native Android domain.

4.4.2 Native domain vs. code-on-demand domain

The user interface is split into two programmatic domains, the native Android domain and the code-on-demand (JavaScript) domain. Both have exactly the same look-and-feel for the user, who is unaware of the underlying technologies used to process his requests and present the UI. The reason for this split is to reap the advantages of both technologies and

use the implementation which is best for the given situation.

Native Android domain

The advantages of the native Android code domain are higher efficiency and flexibility during implementation. Full access to the entire Android framework ensures that all functions of the device like the camera and operating system calls possible. Views developed in this domain (e.g. the Now view, the application's header and the NavBar) are created in Android's XML-based layout language and behavior is coded in a dedicated Java class. Data is fetched asynchronously (using Android's AsyncTask class), via a Hypertext Transfer Protocol (HTTP) client implementation, from the TUManage backend service using RESTful HTTP requests. Parameter data is gathered from UI elements and, if applicable, local system framework calls. Response data is is returned as a JavaScript Object Notation (JSON) data structure [4] and is parsed by a JSON parser, whereafter UI elements are updated with response data and a possibly pending throbber dialog dismissed.

Code-on-demand domain

To handle on-demand code received from the TUManage backend core service an instance of the Android WebView class is embedded in the WallClient application. It is used to render Hypertext Markup Language (HTML)5 documents with embedded JavaScript using the JavaScript engine bundled with Android. When a code-on-demand view is to be shown the WebView control is initialized into the area left of the NavBar and code is fetched from the backend via a HTTP request. The HTML document is assembled on the server using JavaBeans and Java Server Page (JSP) and returned to the WallClient WebView instance which renders it. The familiar WallClientUI look-and-feel is implemented using custom Cascading Style Sheets (CSS) styles, while the UI behavior is implemented using the jQueryMobile JavaScript framework [3]. The framework provides widgets, animations and wrappers for managing multiple sub-views inside a single document. Many quirks related to rendering a smooth UI on touch-based devices are handled by the framework. Lower-level behavior like REST calls to the backend are coded in pure JavaScript using the jQuery library [2] (not to be confused with jQueryMobile) to abstract the browser/WebView Application Program Interface (API). For example, jQuery wraps the browser's more complex XMLHttpRequest object with a $.ajax() call to perform a HTTP call to the RESTful backend service to perform requests or fetch query data. Returned JSON data is parsed natively in JavaScript using a safe wrapper library provided by jQuery. Library functions are also used to traverse the resulting data structure and update the document's Document Object Model (DOM), thereby dynamically updating the UI.

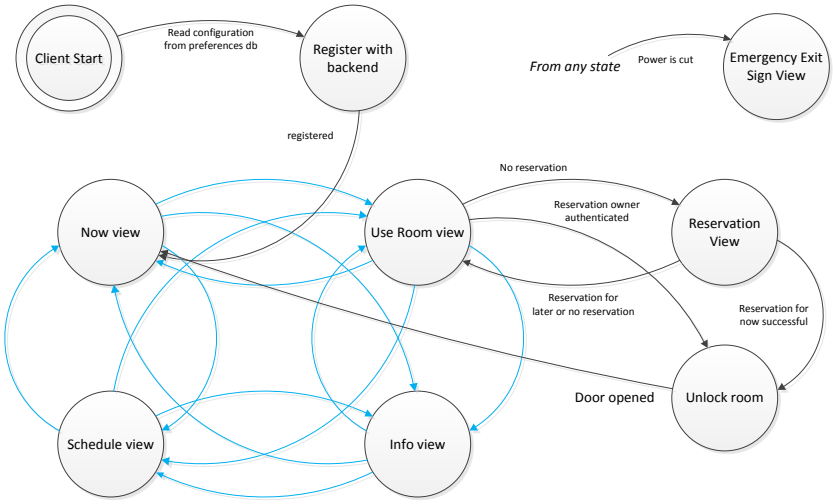

Figure 4.4: High-level state diagram of application state

4.4.3 UI overview

As can already be deduced (hopefully, because that is the design's intention) by looking at the NavBar control described in the section above the UI consists of four main states: Now, Schedule, Use Room, Info. The Now view displays the current state of the room, Schedule allows events to be browsed using a timetable metaphor, Use Room is for reserving the room and possibly unlocking the door. Finally, Info displays information about the room e.g. contact details about the room caretaker and any further, room-specific details. The user can jump directly to any state by touching the corresponding button on the NavBar. Figure 4.4 shows a very high-level overview of the UI state. Each state has one or more views, and each view can have sub-views which are simply hidden UI elements which are shown when needed. The blue state transitions represent jumping to another view via the NavBar controls. The reservations view is reached from the Use Room view when a reservation needs to be placed, starting either immediately to use the room now or at some time in the future. The Unlock Room view causes the room to be unlocked and simply displays a message that the user may now enter the room. It automatically switches to the default Now view once the door is opened so the WallClient is ready for the next user.

4.4.4 Now View

The most important information a WallClient should display while in idle is its room's current state. A passer-by should, at a glance, quickly be able to assess if the room is available or if it is currently in use. Also, participants of future activities in a room should

quickly and easily see if they are about to enter the correct room, where their event (e.g. lecture, meeting) is about to start shortly. The Now view is also the default view and the starting point for the first user interaction. When the terminal detects that no user is present it switches to the Now view to prepare for the next user. In order to generate attention some elements of the view are animated, for example the hint that the room is currently unoccupied can be reserved spontaneously is flashed occasionally as this feature is unknown to new users. Once a user is detected (in future implementations possibly using motion detection on the device's built-in camera) the view is brought back to its "attended" state and the display backlight is set to full brightness. The "attended" sub-view shows details about the room's current and next scheduled event along with information about when next the room is free (and for how long). The times are given in absolute (e.g. 14:30) and relative (e.g. in 32 minutes) values, currently running events are visualized with a progressbar-like area which is filled with color as the event progresses. A button is placed prominently for the user to press should he wish to use the room, it will take him directly to the Use view, see section 4.4.6. The Now view is implemented in the native Android domain, meaning its functionality and presentation are coded in the WallClient application using the Android framework. Data is accessed from the TUManage backend by using remote procedure call (RPC)s to the TUManage backend core services via the HTTP protocol. All requests are RESTful, the data format used for response data is JSON.

4.4.5 Schedule View

The schedule view shows present and future events by making extensive use of the timetable metaphor. Users are used to seeing paper timetables with the week's schedule fixed next to rooms' doors where the WallClient is placed. Therefore it was decided to continue using this familiar metaphor. An example of the view can be seen in figure 4.5.

Grid timeslot scheme

An overview of the current week is presented in a grid formatted according to the TUM timeslot scheme which accommodates a repeating sequence of 90 minutes of event followed by 15 minutes of break. This scheme was adopted by TUM in 2011 to unify room utilization so rooms can be used more efficiently and users (students, staff etc.) have enough time to travel from one event to the next. The exact grid scheme to be used for the display can be changed dynamically by the corresponding backend service because this view is implemented in the code-on-demand domain.

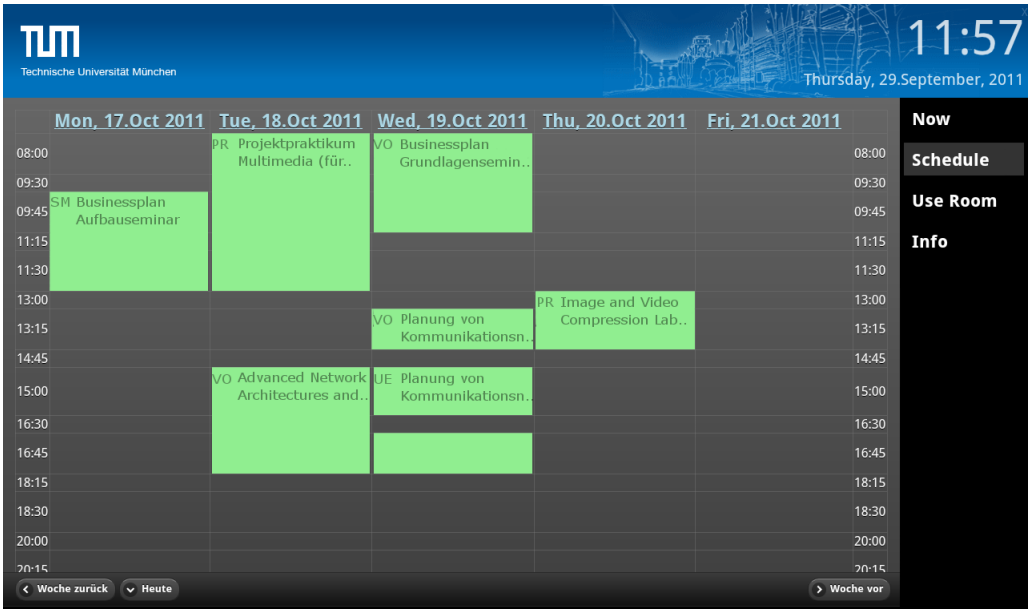

Figure 4.5: The schedule view with some sample events

Code-on-demand domain

In contrast to the native Android implementation of the Now view, when the user requests the Schedule view to be displayed a REST call is made to the backend via HTTP to first fetch the view containing the timetable widget. Once downloaded, the timetable widget class is bound to a backend resource, so it makes another call to the backend to fetch the event data. When the second call completes the data is populated into the grid widget by the class's code. Which code, and with which parameters it is served to the WallClient, can be determined by the backend service at runtime, so different rooms can, for example, show different grids or some other criteria be used for the dynamic behavior.

Info popup

Events are shown as overlays on the grid in the form of colored boxes (depending on the event type) which contain the event's type code (e.g. VO for "Vorlesung", lecture) and the name of the event, space permitting. Longer text is cut off using ellipses (...). Each event overlay is responsive to touch events, this is hinted to the user using an info and hand icon (fig. 4.6). When the user touches the icon or the overlay a dialog is overlayed across the entire view showing the event's full details and any advanced information. The user can close this dialog by touching the "close" button, a timeout has been implemented so the dialog closes automatically when the user steps away from the terminal. Touching any area on the schedule grid not occupied by an event opens a similar dialog asking the user if he would like to place a reservation for the room at the selected timeslot. Finally,

Figure 4.6: Icon hinting touch action to show information

the user can page to the next and previous weeks by using the bar at the bottom of the view. A button is provided to quickly jump to the current week.

4.4.6 Use Room view

When a user wishes to use the room, either immediately or in the future, the "Use Room" shortcut in the NavBar brings him straight to the Use Room view. One of three possible sub-views is now shown, depending on the state of the room.

The room is occupied with a private event

If the room has a current reservation where the privacy setting "lock the door" has been selected the in-use sub-view is shown listing the reservation owner and type or reservation, as well as the remaining time the reservation is in place. If the option "do not disturb" was chosen during reservation this text is also displayed prominently on the view. Buttons are provided for the reservation owner to cancel the reservation or open the door (i.e. when he is using the terminal to gain access to the room during his reservation). In this case the identity of the user needs to be verified so the session is briefly delegated to the Authentication view (see 4.5). Afterwards the user is taken back with the desired options accessible (provided the authentication was successful).

The room is occupied with a public event

Should a public reservation like a lecture or seminar currently take place in the room the extended details about the reservation are displayed, together with a green entry symbol and the notification text that the door is open for event participants to enter the room. Buttons are again provided for the reservation's owner to cancel or modify the reservation, e.g. to lock the door by making the reservation private.

The room is unoccupied

This sub-view tells the user that the room is currently unoccupied and may be used straight away, provided a reservation is quickly performed in 3 easy steps. A confirmation button is provided to take the user straight to the Reservation view, because the room needs to be reserved in order to be used, even if the reservation begins immediately.

4.4.7 Reservation view

The Reservation view is primarily used to place a reservation, beginning either immediately or at a chosen time in the future. In contrast to other parts of the UI the Reservation view's user-interaction style is of the anthropomorphic approach [10]. This approach to human-computer interaction involves designing a user interface to possess human-like qualities. The Reservation view consists of a series of sub-views asking the user information about his reservation requirements. Answering one question brings up the next question, when finally an overview is presented and the user asked to confirm his choices. He can at any time go back or forward to any question using a breadcrumb- and progressbar-like navigation element positioned at the bottom of each view. Questions asked are the following,

- Reservation start time. The user is asked if he would like to start his reservation immediately, in 30 minutes or at some point of time in the future. If the room is currently in use the user is asked if he would like to reserve after the current event.
- Reservation duration. The maximum possible reservation duration is shown (to avoid conflict with another event, else the maximum reservation limit is enforced) and the user can choose the reservation duration in 15 minute steps. The absolute and relative (e.g. "in 23 minutes") start and end times are shown for convenience.
- Reservation use. Next, a list of choices is displayed asking what the reservation is for. This information is used for the Now view information display while the room is in use. Also it is aggregated for statistical purposes for the room administration.
- Privacy settings. Here the user can choose if the door should be unlocked during the reservation time or if only he may unlock the door. In this case the door will automatically be locked for everyone on the outside at 10 minutes after the reservation start time until the reservation ends. The door can, of course, always be opened from the inside. The reservation owner can at all times unlock the door from the outside using the Use Room view's corresponding button and authentifying himself.
- Identification. The AuthView (4.5) is used to determine the identity of the user which is subsequently displayed. An option is provided to correct this identification name in case the user wishes to change it.

Next, a summary of the provided information is presented and the user confirms that the system has gathered his reservation wishes correctly. A RPC is invoked (in the background)

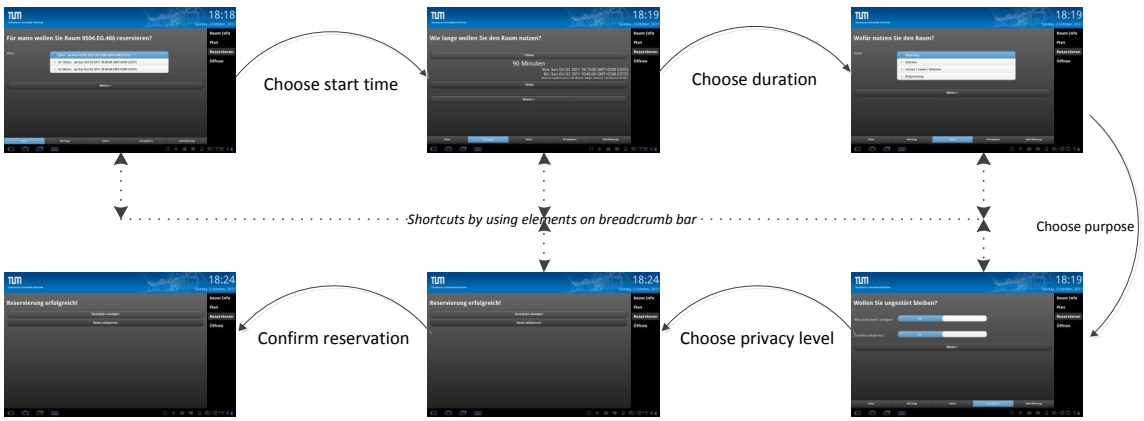

Figure 4.7: Subview sequence while placing a reservation

to the backend's reservation service to place the reservation. While the request is being handled remotely the user is given feedback that the task is being processed by means of a throbber animation (see 4.3.4). Once the backend has processed the reservation request feedback about its success is presented to the user, together with either unlocking the door (if the reservation starts immediately) or the note on how to open the room when the reservation starts (by selecting the "Use Room" navigation). Figures 4.8 through 4.12 show screenshots of the Reservation sub-views of our prototype implementation.

4.4.8 Authentication view

The authentication views are part of the AuthService described in the next section, 4.5.

4.5 Authentication service

4.5.1 Goals

Authentication of the user towards the TUMobile system is a tricky challenge. On the one hand, every TUM user already has a username and login credentials to the campus-wide single sign-on (SSO) service employed at TUM, so it would make sense to use this consistent, secure and well-supported service. On the other hand, security considerations make it difficult to justify requiring users to input their password on TUManage WallClients. Not only could attackers watch users as they enter their password on the easy to observe WallClient touch-screen hanging on a hallway's wall, potential attackers could also install rouge

Figure 4.8: Reserving a room, step 1

Figure 4.9: Reserving a room, step 2

Figure 4.10: Reserving a room, step 3

Figure 4.11: Reserving a room, step 4

Figure 4.12: Reserving a room, step 5

fake WallClient terminals where unsuspecting users would enter their password, believing they are, in fact, interacting with TUManage.

4.5.2 Concept

The AuthService is a service implemented in TUManage where users authentify themselves via their web-enabled mobile phones. Special care was taken to ensure the service is compatible with virtually all smartphones available by using standard web technologies omnipresent in all smartphone platforms. Client authentification is performed using a trusted and protected terminal, the smarphone. Server authentification is performed via standard Internet public key infrastructure (PKI).

4.5.3 Sequence

Please refer to figure 4.13 for a communication sequence diagram and figures 4.14 through 4.18 for the sequence of UI views presented to the user. The large screenshots are from the WallClient while the smaller, portrait-orientation ones are from the user's smartphone. When authentification is required by TUManage the AuthService displays a Quick Response (QR) code on the WallClient containing a special, publicly accessible Internet Uniform Resource Locator (URL) of an AuthFramework resource hosted on the TUManage backend server. The URL contains a newly generated session ID in its query parameter payload. The user is asked to scan the code using his Internet-enabled smarphone which

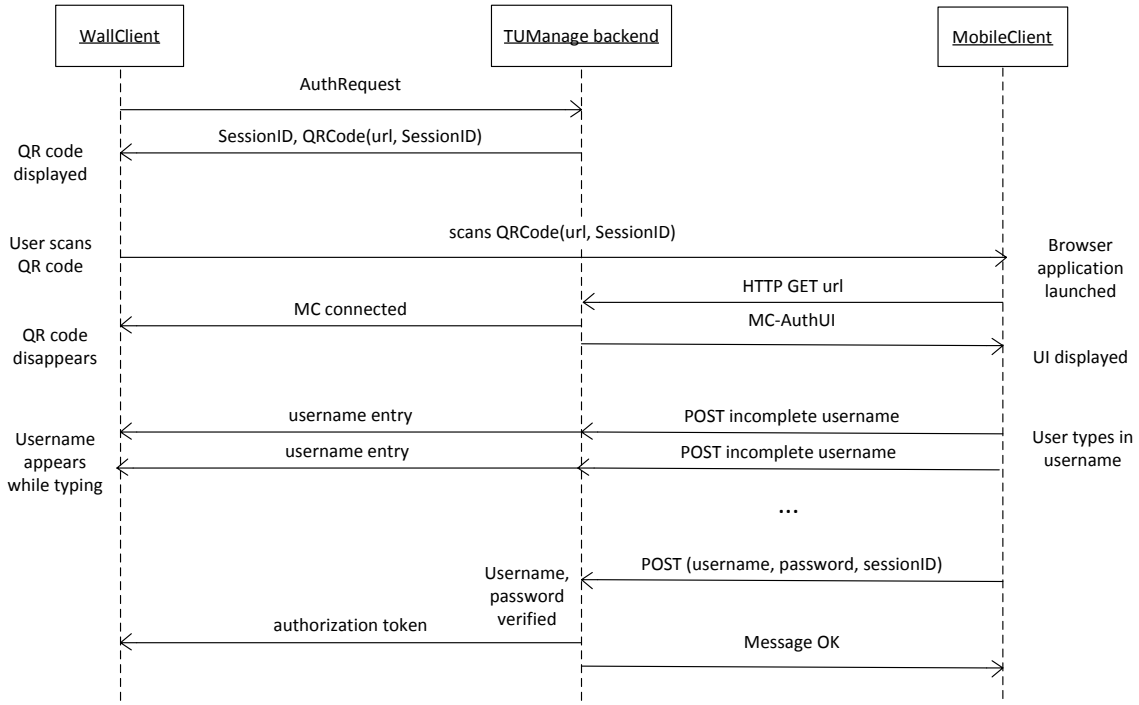

Figure 4.13: Sequence diagram of user authentification using MobileClient session through AuthService

will follow the standard Hypertext Transfer Protocol Secure (HTTPS) URL by opening the phone's web browser. Once the connection is established the backend will serve an instance of the Mobile Client (MC) AuthUI, a JavaScript enabled HTML5 web application. Simultaneously, a message is sent to the WallClient to stop displaying the QR code to minimize the exposure of the security-sensitive QR code. The sessionID sent on the QR code's payload is now tied to the MC AuthUI instance running on the smartphone. On the WallClient the user is now presented with a message to continue the authentication procedure on his smartphone. Here, a field for username and password are provided, where the user enters his details. While entering his username, the character input is relayed to the WallClient via the backend service so the text appears on the WallClient while the user is typing it into his smartphone. This assures him that his smartphone really is paired with TUManage and the WallClient session and some attacker has not scanned the QR code over the user's shoulder to "steal" the session. The user's credentials have now reached the server through a secure channel and can be verified using a credentials store or a Lightweight Directory Access Protocol (LDAP) call etc. Once the user is successfully authenticated his sessionID is marked as valid and he can continue his now authenticated session on the WallClient. A result message is displayed on the smartphone that the session will now continue on the WallClient and the smartphone is no longer needed.

Automatic sign-on

If the user wishes, he can permanently bind his TUManage identity to his smartphone, for a maximum duration of 1 month. In this case, a cookie containing a TUManage authorization token is stored on the smartphone. In future, when a WallClient session needs to be authenticated, the user simply needs to scan the QR code presented. The smartphone's initial GET request to the backend AuthService resource will contain the cookie containing the identity token which is matched by the AuthService's token store in the TUManage backend. Instead of serving the MC AuthUI the session is authentified immediately and the user does not need to enter his username and password.

Security mechanisms

To sum up, authentification of the user towards TUManage is done using a username/password pair. This account is either registered directly in TUManage's backend datastore or it can be verified using the TUM SSO service via a dedicated LDAP interface between the TUManage core service and the SSO server. In our prototype implementation the simpler internal authentication using the TUManage credentials store was implemented. Authentification of the server towards the user is done via Secure Sockets Layer (SSL)/Transport Layer Security (TLS) certificates using the same principles as described in section 4.6. The user simply needs to ensure that the URL displayed in his smartphone's browser after scanning the QR code is really that of TUManage, and that

Figure 4.14: Step 1: QR code is displayed on WallClient

no certificate errors are displayed by his smartphone. The TUManage server certificate is validated by the chain of trust provided by the PKI trusted by the smartphone's browser vendor.

4.6 Security

With the TUManage system deployed at multiple locations within a building it needs to be interconnected using easily available network infrastructure. Care has been taken to ensure these basic security services: Confidentiality, integrity, authenticity [27]. Implementation of these services is achieved using the SSL/TLS protocol which provides server authentication, data encryption and data integrity functionality.

4.6.1 Implementation

On the WallClient side the SSL/TLS stack integrated into the WebView is used for the JavaScript domain, for the native domain the org.apache.http.impl.client.DefaultHttpClient bundled with Android is used. The TUManage backend service makes use of Apache httpd's SSL/TLS stack, see 5.4.2. Certificates for the backend service do not need to be explicitly imported into every WallClient device if the server certificate is registered by a Certificate Authority (CA) recognized by a root CA included in the Android framework. In this case the chain of trust provided by the PKI trusted by the Android platform reaches the server certificate without interruption. This is the case with our prototype implementation where the TUM CA signed the TUManage server certificate, whose certificate was in turn signed by a CA

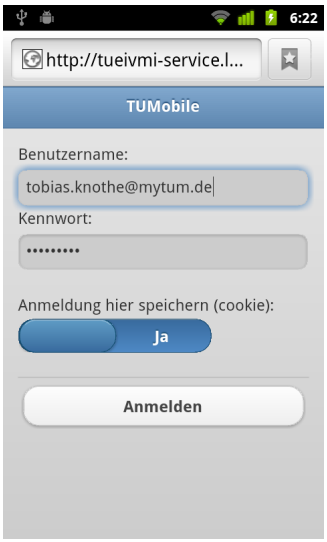

Figure 4.15: Step2: QR code scan opens MC AuthUI

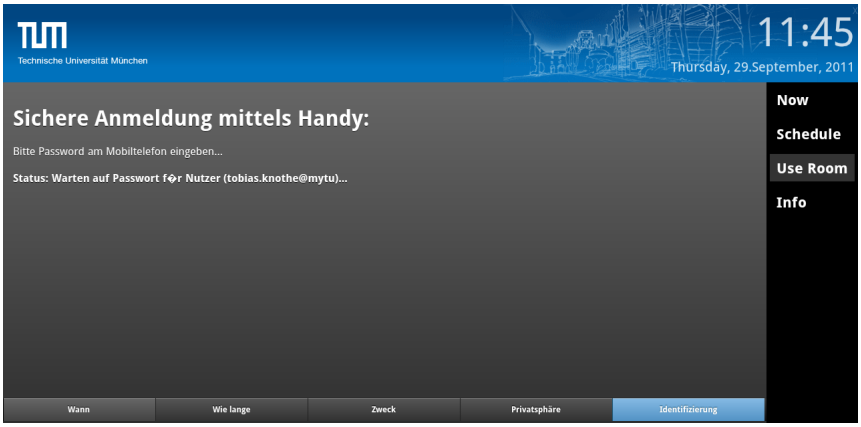

Figure 4.16: Step3: Username is entered and appears on WallClient while the user types on the smartphone

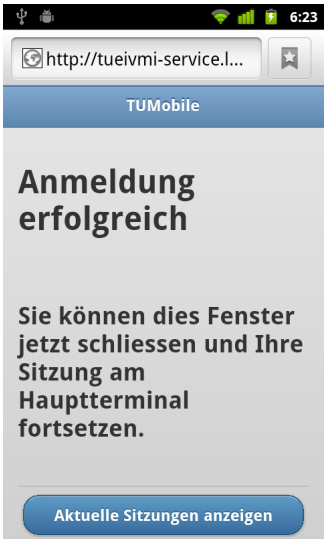

Figure 4.17: Step 4: Authentication succeeded displayed on the smartphone

Figure 4.18: Step 5: Authentication success displayed on the WallClient, session is now authenticated

trusted by the Deutsche Telekom Root CA, which is recognized by and included in the Android platform operating system.

4.6.2 Mechanism

When a SSL/TLS connection using the HTTPS protocol is initiated the connection is established using a handshake protocol. The client and server negotiate the strongest algorithms from their CipherSuites (list of available cipher mechanisms for e.g. hashing and encryption) and exchange certificates. These certificates may be validated by contacting the issuing authority to ensure the certificate is valid and has not been revoked. Next, a session key is generated using the asymmetrically encrypted exchange of a random number. The session key is used to encrypt the session, which is form now on held using the HTTP protocol.

4.6.3 Client authentification

The use of a server certificate is standard with HTTPS but this only authenticates the server to the client (and helps protect the session key). Client authentication is where the client additionally sends its certificate to the server so the server can authenticate the client. Because there is considerable overhead to deploy this form of authentication (all clients need a certificate signed by a CA recognized by the server) our first prototype implementation uses a different mechanism for client authentification: The HTTP Digest authentication scheme as defined in RFC 2617 [11]. Here, login credentials or a Pre-shared Key (PSK) from the client are verified by the server. A hashing algorithm and the use of a number only used once (nonce) prevent the credentials from being sent in plain-text on the HTTP layer and make cryptanalysis difficult. Plain-text authentification would also be possible as the WallClient has theoretically already authentified the server before sending the password on the HTTP layer but it is still possible that HTTPS is accidentally switched off during future maintenance of the TUmanage system. Then the digest authentication would serve as a fallback to prevent plain-text passwords being sent over the unsecured network connection.

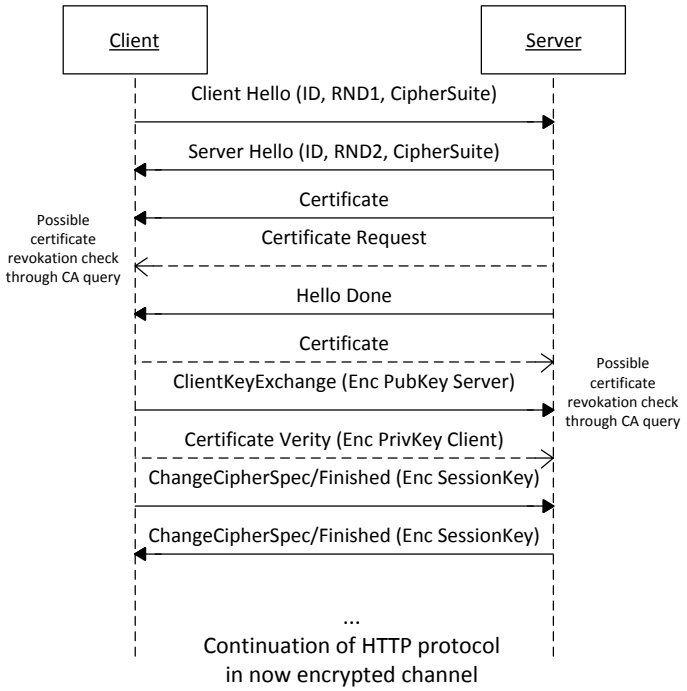

Figure 4.19: SSL handshake sequence diagram (Source: [27])

Chapter 5

Backend service

5.1 Goals

The counterpart of all WallClient displays is the central TUManage backend core service. If forms a centralized entity responsible for keeping state of the current room data. Instead of storing data like a room's state or the room reservations on the WallClient itself it is managed by the backend which holds the master instance of all data. The advantage of this approach is that all data is available across all devices and concurrency is guaranteed. An example would be that a building manager could see an overview of all rooms' states on a special administration interface (either on the same hardware as the WallClients or a web interface). Another example would be when rooms with multiple doors have a WallClient at each door. When state is changed using one WallClient this is immediately propagated to and visible on the others. When RoomControl clients are used to monitor a room's environment's state this data, too, is aggregated at the core services and available on all devices and clients.

The backend core service provides a common interface to all WallClient instances and handles their requests via a REST based request-response model (described in section 5.2.3 below). State and other information is managed transparently and centrally, while ensuring data concurrency and consistency.

Apart from handling requests by WallClients, the backend service also contains services for handling requests by external clients using the AuthFramework (see section 4.5), for example, when a user authenticates himself to TUManage using his smartphone.

5.2 Architecture considerations

A couple of core decisions were taken while implementing the TUManage core backend. For one, a client-server based approach was chosen, where the core backend takes on the role as a server and the WallClient devices are the clients. A Service Oritented Architecture (SOA) was implemented using the REST style, resulting in so-called "RESTful" resources offering

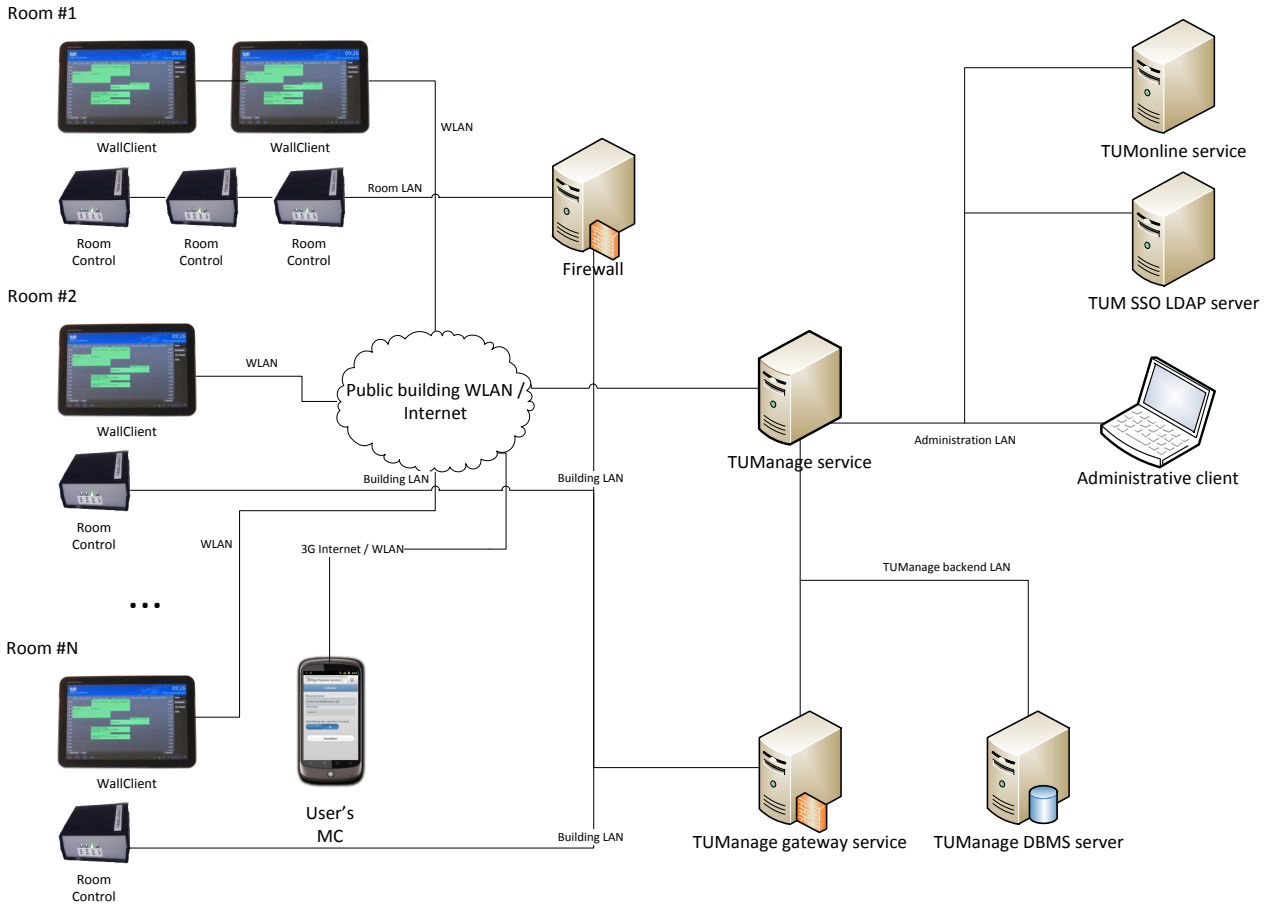

Figure 5.1: High-level architecture of core backend in global context of the whole system

services in form of methods invokable by clients. This section illustrates the core high-level architecture of the backend, while sections 5.4 and 5.6 describe the platform used and the implementation. Section 5.5 takes a detailed look at the actual services offered.

5.2.1 System Architecture Overview

This section provides a high-level overview of the TUManage core backend in the context of the global TUManage system.

5.2.2 Service oriented architecture (SOA)

SOA is a collection services with a loose coupling and dynamic binding between services [26]. It relies on service orientation as its fundamental design principle. A service is a

well-defined, self-contained function which does not depend on context or state of other services. If a service presents a simple interface that abstracts away its underlying complexity, users can access independent services without knowledge of the service's platform implementation. Services manage their own data and present a coherent and consistent interface to service requesters.

One main reason for adopting a SOA is the efficient interoperability and interconnection of systems running on different platforms and written in different languages. Also, services can be re-used, making implementation very efficient and easily scalable. For example, the same service is invoked in the background by WallClients to query room events when both viewing room info and when reserving a room. Another example would be the AuthService and the BarcodeService. They are used by different parts of the TUManage system at different times and in different context, yet the same service is requested.

TUManage uses web services as the approach for building a SOA based on web technologies, adopting the REST style.

5.2.3 REST

REST stands for Representational State Transfer and is a term introduced by Roy Fielding in [16]. REST is an architectural style for distributed systems, derived from architectural styles like "Client-Cache-Stateless-Server" and "Layered-System" and subjected to an additional set of constraints. The constraints considered when specifying the TUManage backend are briefly described in this section, as well as their implication on TUManage's design.

Client-Server

The client-server style, widely used in network-based applications, defines a server component as a reactive process triggered by client who make a request. With TUManage, requests to the central TUManage backend server originate from any one of the WallClients. There is a common interface defined for communication between clients and server, in this case HTTP exposed resources (realized by Java Servlets) with parameters and response types shown in Appendix A. On the WallClient clients communication is abstracted either using the HttpRequest/HttpResponse classes for code running in the native Android domain or an Asynchronous JavaScript and XML (AJAX) wrapper for those parts running in the JavaScript.

Stateless

The requests to TUManage are stateless, so no information about a transaction is stored by the server until the transaction completes. Each request from client to server must contain all the information necessary to understand the request and cannot take advantage of any stored context on the server [16]. As soon as a request has been answered the communication session ends. Advantages of this constraint include better scalability, reliability, and generally a simpler, easier to debug communication.

Cache

Although caching of requests from the WallClients is not performed in a normal TUManage use-case, the possibility caches present in the HTTP data transfer path are still possible. While the infrastructure connecting the WallClients to the TUManage backend services is known (it usually belongs to the building administration, in our prototype's case the campus' wireless LAN provider) it is possible that e.g. transparent HTTP proxies are integrated into the infrastructure at some point in the future.

Uniform Interface

The overall system architecture is simplified and visibility of interactions is improved by using a general and uniform interface between components. The implementation of the interface is decoupled from the services it provides. The TUManage core service, being a request-based RESTful system, uses the standard HTTP GET/POST interface. Resources are identified by Uniform Resource Identifier (URI)s and state representations are usually returned in JSON [4] or XML, or as a human-readable HTML document.

Layered System

Component behavior must be constrained in such a way that each component cannot "see" beyond the immediate layer with which it is interacting [16]. For a client it is transparent which server is used for the request, or if the request has traversed a proxy or cache or load balancer. For example, within TUManage the backend's Structured Query Language (SQL) database is completely hidden from from frontend WallClient transactions, who never see the entire data or how it is stored. Also, clients need not know which server is handling their request. TUManage could be scaled by running multiple instances of the backend services on different application servers and load-balancing requests among them. A common SQL Database Management System (DBMS) server or a DBMS cluster would then ensure consistency of state throughout the system.

Code-on-demand

In the code-on-demand style, a client component has access to a set of resources, but not the know-how on how to process them. It sends a request to a remote server for the code representing that know-how, receives that code, and executes it locally ([16]). WallClient functionality is extended by downloading and executing code in the form of JavaScript when requesting resources outside of the Android native code domain. This directly results in less deployment overhead when adaptations need to be propagated. This functionality is not implemented at the client, it is downloaded upon request. As an example, when a request to reserve a room is made by the WallClient, the exact process is not yet known to the WallClient engine. The TUManage backend serves the code required to complete the user's reservation process dynamically, in the form of JavaScript, which is executed in the context of the UI the user is acting upon. Customization can therefore be served on demand and be changed dynamically (for example, during special events like seminars the user could be required to state if he is a seminar guest or a member of the normal staff).

5.3 Summary

The main reasons for using REST to implement SOA are simplicity, robustness and scalability. Once many clients interact with the server and request service a considerable overhead would be generated if state had to be stored for each client or client session. Both the services and the system itself would become increasingly and unnecessarily complex and prone to bugs, especially during maintenance and scaling in the future. Debugging and overview of the system internals are greatly enhanced.

5.4 Platform considerations

5.4.1 Servlet container

The TUManage core services are implemented as a JavaEE [24] web application hosted on a platform (called container) implementing the Servlet 2.5 and JavaServer Pages 2.1 specification. Apache Tomcat version 6.x [14] was chosen because it implements the required specifications while still being very light-weight. Version 6 has been proven to be stable and ready for productive use.

5.4.2 Security

An Apache httpd server [1] runs alongside Tomcat to handle incoming requests from remote clients instead of Tomcat itself accepting incoming connections from the outside. The reason for running httpd as a proxy between Tomcat and remote clients is in order to utilize httpd's SSL processing. HTTPS is enforced on all incoming connections (they should already be requesting a HTTPS connection in the first place) so that communication is encrypted and the server is authenticated to the client. A secure communications channel is thereby established over a potentially unsecure network, as a protection from eavesdropping and man-in-the-middle attacks. httpd connects to Tomcat via the Apache JServ Protocol (AJP) connector using the AJP protocol, a Transmission Control Protocol (TCP) based binary protocol created specifically for this task. In Tomcat it is implemented natively and on the Apache httpd server the mod_tomcat server module is used. AJP features high performance, reuse of a connection for multiple request/response cycles and access to the SSL details like cypher suite and client certificates from the httpd proxy. Security for the communication with the WallClients, which are at the other end of the encrypted connection, is discussed in detail in the section on WallClient security, section 4.6.

5.4.3 Database management system

Data in the TUManage core service is persisted exclusively in the DBMS due to its superior transaction management and mechanisms for ensuring consistency and concurrency. No data is stored directly inside the JavaEE application and all persistence classes are wrappers for invocation of procedures on the DBMS. The reason for the strict separation of code and data is code clarity, for one, but also it allows the system to be scaled easily, see 5.4.4. The open-source MySQL community edition server [6] was used because it fully supports all required features, meets performance expectations and it available free of charge. Connection to the database abstraction layer is via Java Database Connectivity (JDBC) through MySQL's own JDBC driver (called MySQL connector/J).

5.4.4 Server host

Requirements for the server platform were simply the ability to run the Tomcat and MySQL stacks. A Linux flavor was chosen due to installation and management (updates, patches) simplicity as well as security. It was decided to already make provisions for scaling the system by setting up two servers, currently one for the public-facing web application and one for running the MySQL DBMS. Unix cron was used to trigger automatic recurring tasks by programmatically invoking special, non-public servlets at specified intervals. This is used for periodic maintenance, data synchronization with other systems and state synchronization with TUManage RoomControl clients, if applicable. When scaling server

hosts care should be taken to run the cron service on only one server, otherwise jobs would run multiple times, which is unnecessary in terms of resource consumption.

Scaling

In future, different parts of the web applications services could run on different servers, or the entire environment is cloned across multiple physical machines while dispatching incoming requests using a load balancer. The DBMS would, by design, ensure concurrency and consistency of data, even if it, too, is distributed across multiple machines in a master-slave DBMS cluster (available from MySQL and other DBMS vendors).

5.5 Services offered

Services are provided by the TUManage core backend as a set of RESTful resources identified by their URIs. Abstract resource URIs end with .resource, they are RESTful and can be fetched or manipulated using HTTP methods. Direct services are addressable via their URIs ending with .do. This section attempts to highlight the implementation of some important services offered by the TUManage core backend. As the resources fall into different organizational packages (both logically and in URI path) they are explored in the following paragraphs according to package. A full list of resources, methods, parameters and return types can be found in the code documentation accompanying this thesis' practical part. An excerpt can be found in Appendix A.

5.5.1 Administrative resources

Administrative services are in the /admin/ URI namespace and are used to administer the TUManage platform. Elevated privileges are required to access these resources. They include methods to, for example, kick-off a synchronization of room and event data from the TUMonline system. A list of the most important ones can be found in table A.1.

5.5.2 Room management resources

These resources represent the room being managed (Room) as well as reservations and events taking place in the room (RoomEvent). Methods exist to manipulate reservations (query, create or modify RoomEvents). The RoomLock resource is a little bit more abstract, it is used to unlock the room's door by successfully (i.e. with the proper authorization and at the proper time) invoking its GET method. The URI namespace of these resources is /manage/*, please see table A.2 for some examples.

5.5.3 WallClient resources

The WallClient resources are used by the WallClient terminals' GUI to download code for viewing and/or manipulating room or event resources. Specifically, they are used by the JavaScript domain of the WallClient application, and TUManage serves JavaScript code inside HTML5 documents. The resources are in the URI namespace /wallclient/*, a few examples can be found in table A.3.

5.5.4 AuthService resources

Resources part of the AuthService can be found in the /auth/* namespace. They are used for e.g. creating auth sessions or to authenticate a user by binding him to an opened session after verifying his identity. The QR code scan performed by the users points his smartphone to the AuthenticationClient.resource which causes a code download of the JavaScript/HTML5 authentification client. AJAX is then used to interact with the other AuthService resources by invoking other AuthService resource's me. Please see section 4.5 for a description of the AuthService.

5.5.5 Barcode resources

The barcode resource can be found at /resources/Barcode.resource. It is used to generate a 2D QR code containing the payload encoded in the GET request's encdata query string. In the prototype implementation Google's chart API is used to render the barcode containing the payload assembled by the Barcode.resource barcode factory class.

5.5.6 Crond resources

These resources are not accessible from the outside network connection, rather their methods are invoked automatically from the crond service (a job scheduler in Unix operating systems) running on the local server via the /resources/cron/* path. Crond will, at intervals set in the crond config file, periodically invoke certain servlet via a local HTTP call to perform periodic administrative tasks. One important task is the automatic unlocking of rooms with public events taking place. The unlocker service will check if any locked rooms currenlty have public events of have public events due to start in less than 10 minutes and unlock these rooms via their RoomControl (see 6) systems. The crond is also used to synchronize data with external systems (see 5.7.1).

5.6 Implementation details

5.6.1 MVC and Servlets

Controllers

The TUManage backend core services are implemented using the Model-View-Controller (MVC) design pattern [17]. Each package offering services has one or more servlets, acting as controller class, implementing the HttpServlet abstract base class. This allows the servlet container to dispatch incoming web requests to an instance of the appropriate servlet, using a thread from the application container's (Tomcat) thread pool. The servlet acts as the controller in the MVC pattern by performing the following steps. First, the request's parameters are processed and checked for missing or invalid data. Then, actions are performed on an instance of the appropriate model. Finally, the view is invoked to present a response back to the requester.

Models

Models are objects abstracting behavior, data and state. They are acted upon by requests to the service by means of the controller servlet. Models are instantiated by the persistence layer's store or factory classes. Operations on the model are translated to Create Read Update Delete (CRUD) operations by the database abstraction layer (see 5.6.2) which are passed to the DBMS.

Views

Views are used by the controller to present data or results back to the requester. The format of the response can be either in JSON or HTML/JavaScript. A JSON view is instantiated by invoking a JSON rendering engine for creating machine-readable output from a model object. This is used primarily by the WallClient native Android domain and the JavaScript domain's AJAX RPC calls). HTML/JavaScript is created by dispatching to a JSP page in the /WEB-INF/views/ directory. This is used primarily by the WallClient JavaScript domain to serve combined code and data. It is also used by the AuthService's MobileClient service (with JSON-based AJAX calls in the background).

5.6.2 Database abstraction layer

In order to separate database connectivity implementation to the controller fetching/persisting model objects a database abstraction layer has been implemented, called the data store. Access to the underlying DBMS system is supported by the open-source

myBatis object-relational mapping (ORM) framework [5]. It was chosen over other commonly used ORM frameworks like Spring [7] or Java Data Objects (JDO)/Java Persistence API (JPA) implementations due to its light-weight support for easily obtaining full control over SQL commands used in the background when communicating with the SQL DBMS. Only with a fully customizable interface to the DBMS is it possible to support any given DBMS schema or to design one's own, which was favored in trade-off for higher initial implementation effort. myBatis essentially provides a mapping of JavaBean objects to relational database queries in SQL, and vice-versa, using pre-defined mappings taken from Java annotations or loaded from XML descriptor files. This mapping supports CRUD and can therefore be done in both directions, so a database read can output a populated JavaBean, or a JavaBean's data can be converted to a database create/update/delete. Also, a JavaBean may be used as a parameter for a database query returning another JavaBean. This flexibility allows complex SQL queries to be abstracted behind methods the datastore exposes, using only object oriented principles in the actual code, keeping it free of SQL.

5.6.3 Database schema

Figure 5.2 shows the database schema used by TUManage, in Chen's notations [19], [15]. The schema implementation can be created from scratch on a MySQL DBMS server by running the /dbms/*.sql scripts found in the release package of this thesis' practical part. The room entities (rooms, roomTypes, roomPurposes) are used to store data about the room itself. roomEvent is data about an event taking place in a room (in a past, present or future point in time). The auth* entities are used by the AuthService while the userID can be used by any service to describe a particular authenticated user, as is done by the room and roomEvent entities. Their authUser entity refers to the authenticated user and authSession is any (both authenticated to a user or unauthenticated) session with the AuthService. Sessions can be reused once authenticated, up until the expiry time, if the corresponding service allows this.

5.7 Interface to supporting systems

In order to integrate tightly and seamlessly into the user's environment TUManage is required to interact with other existing systems. For one, the existing room management system employed at TUM, TUMonline, is currently used to manage the lecture halls. Many smaller seminar rooms are managed using s selection of paper sheets, Microsoft Exchange or similar methods.

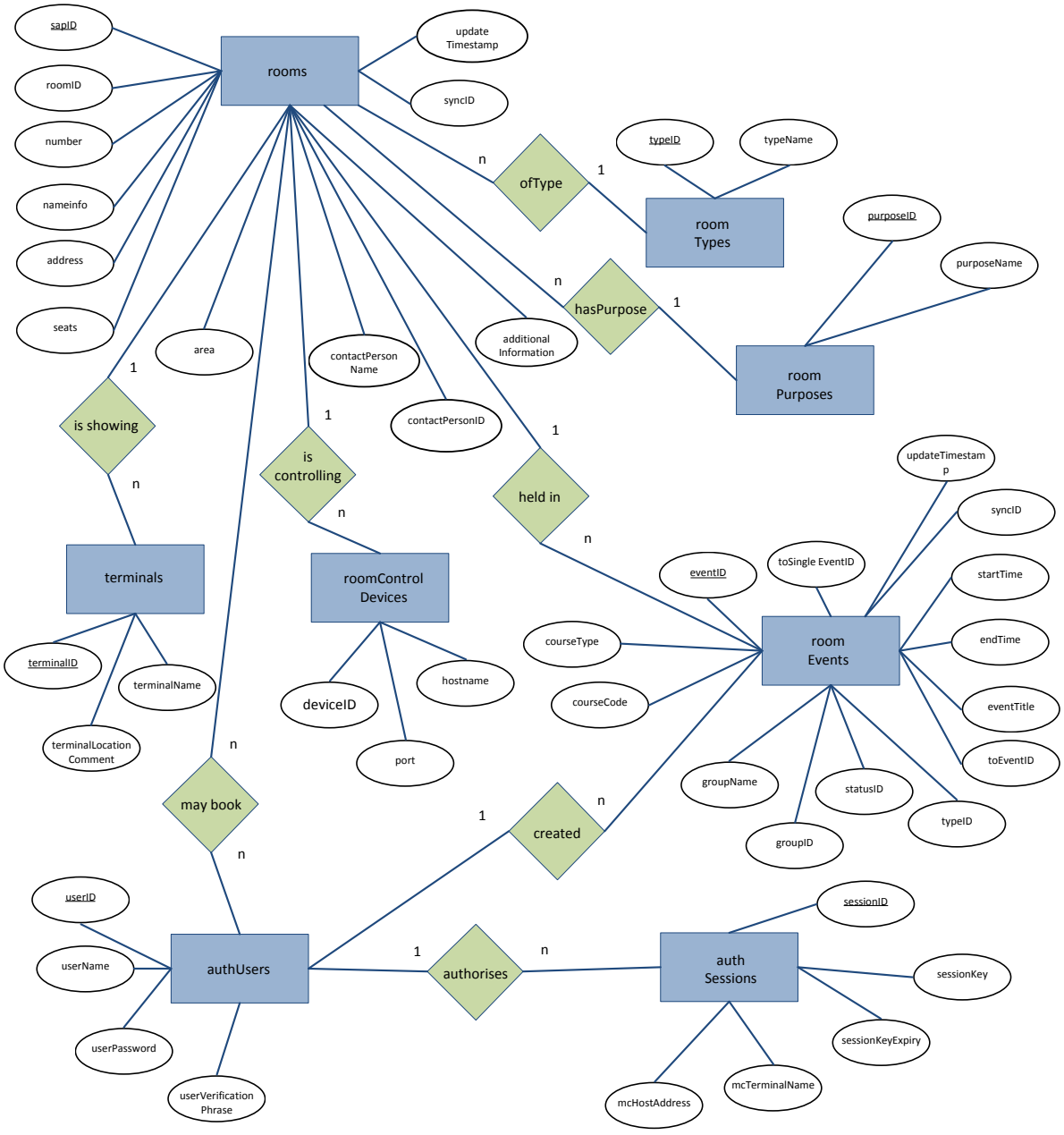

Figure 5.2: Entity-relationship model or implemented database schema

5.7.1 Interface to TUMOnline

TUMonline was developed by the TUGraz and is now maintained by their IT department under the name of Campus Online. TUMonline is a web-based complete university management system which also provides for the management of rooms by univerity staff. Currently it is the official tool of choice at the TUM and it is compulsory to manage lecture halls with this tools. As TUManage is used to manage all kinds of rooms, including lecture halls, it needs to synchronize with TUMonline.

Sync strategy

As TUMonline is only used for very static, seldom changing events like lectures, which recur on a weekly basis, a one-way sync from TUMonline to TUManage was evaluated as sufficient. TUMonline should explicitly not be used to reserve rooms for one-time seminars or ad-hoc meetings. TUManage should take care of these events while importing the static events like lectures in regular intervals. From evaluation of TUMonline a sync period of once per day seems sufficient as the event date itself is not updated very much often. Still, the prototype implementation of TUManage synchronizes once every hour in order to study system performance.

Data exchange

TUMonline offers an authenticated data export mechanism for room and event data via a HTTP resource. The URI is queried with parameters like the room's SAP ID (SAP is the commercial asset management system used at TUM), the start and end dates for the query and an authentication token. It is also possible to query for a list of room SAP IDs by specifying an administration's oraganization ID. Returned data is in XML format utilizing a schema specified by Campus Online.

TUManage sync mechanism

As can be see in figure 5.3 the sync process is initiated by the controller servlet which supervises the process writes transaction information to the log. Parameters from the query are assembled and passed to the QueryEngine, which creates an authenticated HTTP connection via a query to the TUMonline gateway service. The XML data received in the response, an XML document in the RCMv1.0 schema [], is passed on to an instance of the XML parser.

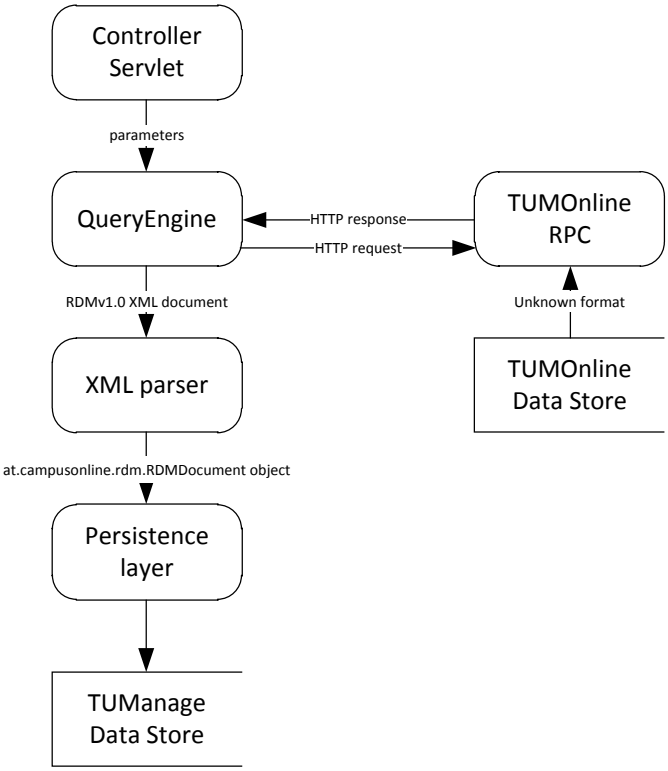

Figure 5.3: Data flow diagram of one-way sync with TUMOnline

XML Parser

To implement this parser the Apache XMLbeans [] framework was. It generates JavaBean-compliant Java classes from the XML schema obtained from Campus Online. Special parser methods are attached to data classes which wrap calls to the XMLbeans library factory classes. Invoking the appropreate method on a new instance of such a root object with XML stream data will automatically populate the object with the entire data structure contained in the XML document. The given XML schema is, in essence, "translated" to a Java data structure consisting of primitives, classes, Lists and Maps. The instance's data is populated form any XML stream adhering to the Schema.

Persisting the data

This JavaBean is dropped to the persistence layer implemented as a store where the myBatis ORM creates a set of SQL "update" and/or "insert" queries sent to the DBMS for persisting the data. Changes are committed by the controller servlet if there are no errors and the changes are taken into effect.

Automatic sync

The process descripbed above can be kicked off by manually invoking the desired sync controller servlet. The process is in addition also automated using the TUManage server's crond service. It will, at intervals set in the crond config file, periodically invoke the sync controller servlets which will perform the sync. Log data is written to disk for debugging in the case of errors.

Chapter 6

Interacting with the environment

6.1 Introduction

A crucial part of the TUManage system's usability and transparency is its capability of interacting with the user's environment while being noticed as little as possible. For our first prototype installation it was decided to control and monitor a room's door lock. A module, called RoomControl, was designed and built for this purpose, a sensor and an actuator were attached to detect the door's state and control its lock. This chapter attempts to describe the RoomControl module design and implementation, both for the hardware and the embedded software parts. At the end a brief look at the prototype installation is given.

6.2 Requirements

From the usability consideration for the TUManage system as a whole the following key requirements for the RoomControl module were distilled.

Transparency - A user should not need to worry about how to operate the lock system, i.e. it should not require user intervention to operate. It should be clear to the user wheather he can directly enter the room or if additional action is required, like authentification with the system. Once his identity has been determined there should be no further action required by the user and he should be able to enter the room. The user should not be confused or surprised by the lock system's operation. An example of this would be that it should always be clear in what state the lock is in, otherwise a user could falsely assume that the door is unlocked and be presented with an unmovable door handle. This requirement is addressed by both the WallClient GUI (see section 4.4) and the door actuator (see section 6.6).

Safety - In order to comply with building safety regulations a user wishing to open the door from the inside should always be able to do so, even under unforeseen circumstances like emergencies or power failures. No extra precautions (like a backup battery etc.) had

to be taken as all doors controlled by the system continues to comply with site regulations. They have unlockable handles on the inside which bypass the lock mechanism implemented by the RoomControl.

Security - A room's door and its lock are a vital part in securing its content against unauthorized use and theft. Connecting this lock to a complex system via network infrastructure can create numerous new opportunities for a potential attacker to gain control of the room's lock or access its state. This is addressed in its own section, 6.5.

Cost - The RoomControl module's total cost needed to be kept at a minimum while still remaining flexible while the system is in the prototype phase. Throught the use of a value-for-money rapid prototyping platform (see 6.3.2) as well as standard, off the shelf components (see 6.3.3) the total bill of materials (BOM) could be kept well under the project's given constraints. Please see the parts list in Appendix B for the complete list.

Considering the above requirements two prototypes were designed and implemented. Prototype 2 was installed in one of the campus' lecture halls for demonstration and testing. Below follows a detailed description of the RoomControl's components and the sensors and actuators attached in our prototype installation.

6.3 RoomControl module

6.3.1 Overview

The RoomControl control module provides the software and hardware interface for other parts of the system to interact with the room environment, notably the room's door and its lock. It should therefore be able to detect the door's state (i.e. open or closed) and be able to lock or unlock it.

6.3.2 Controller microprocessor

While there are many possibilities when choosing an embedded microprocessor for the task at hand, the best fit was evaluated to be the Arduino [12] platform. Due to its low cost and open-source rapid prototyping environment, plus the availability of many libraries and hardware extensions, it was found to be the best suited platform for the requirements at hand. Other solutions like embedded devices running an embedded Linux were evaluated but turned down due to the higher hardware costs involved. The Arduino variant used for the RoomControl is an Arduino UNO, which is a Printed Circuit Board (PCB) based on the Atmel ATmega328 microprocessor chip. The board hosts the microcontroller, some circuitry for power supply and communication with the development environment (a FTDI USB-to-UART driver chip) and pin headers for exposing the 14 digital input/output plus

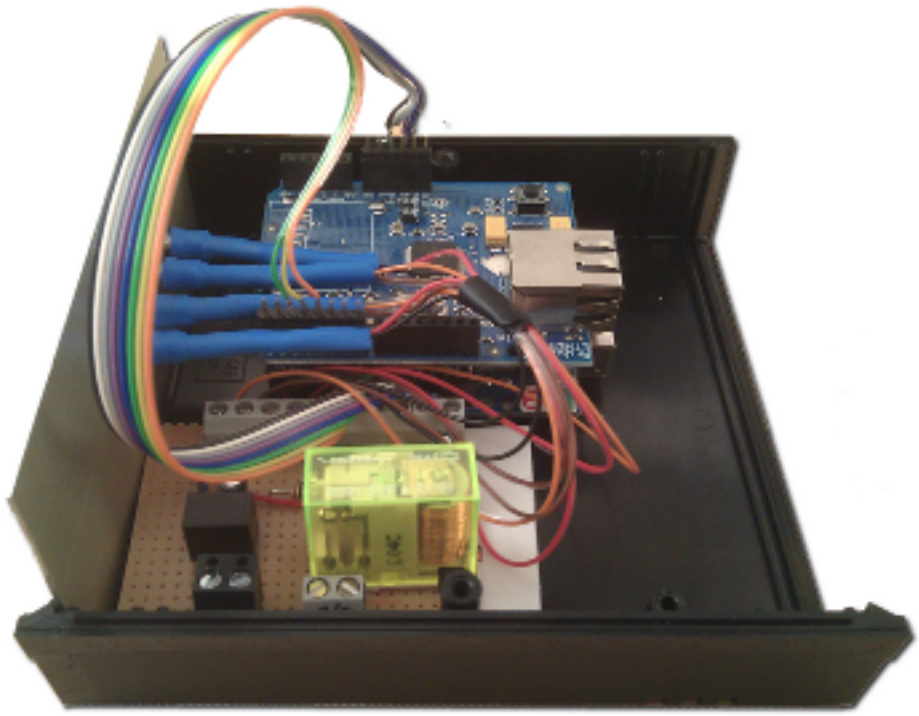

Figure 6.1: RoomControl box (opened lid) with prototype 2 circuit board, Arduino and Ehternet shield

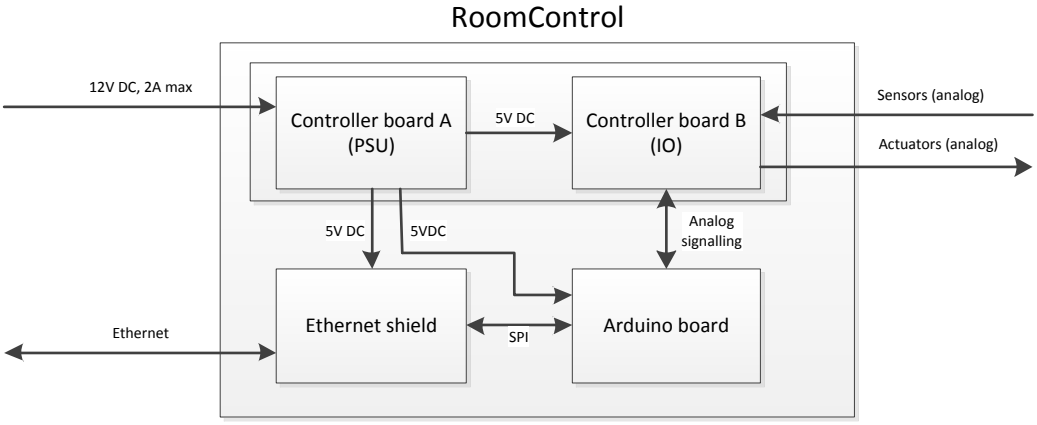

Figure 6.2: Architecture of the RoomControl

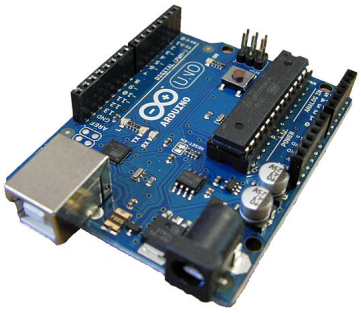

Figure 6.3: Arduino UNO (Source: Wikimedia Commons, CC-license)

6 analog input pins. Schematics for the Arduino UNO board can be found in Appendix C. The Arduino board can be programmed using a C++ like syntax in a custom IDE available from the Arduino project website [12]. The programming language syntax and the libraries used are based on those of the open-source Wiring platform [13]. Standard libraries are available for hardware abstraction and it is possible to import or design 3rd party libraries. A compiler and bootloader are also part of the platform and transparently integrated. - To compile code and launch it on an Arduino board attached via Universal Serial Bus (USB) only two clicks are required.

6.3.3 Controller IO and PSU board

The hardware connectivity module is shown in figure 6.5 and is a custom-made circuit board which has two responsibilities. Firstly, it connects the Arduino microcontroller board to the sensors and actuators of the embedded RoomControl system (the Input/Output (IO) part). The second function of this circuit board is to provide a robust and stable power supply (the Power Supply Unit (PSU) part).

Power supply

Because the RoomControl and its attached components have individual demands on power supply, special attention needed to be given to a single, robust power source. Table 6.1 shows the power supply requirements for the individual components.

As can be seen in table 6.1 the power supply requirements for the RoomControl are quite different from component to component but the common denominator seems to be that both a 5V and a 12V supply should be sufficient. The Arduino board itself has built-in regulators for its internal operating voltage of 5V and 3V but, as it turned out during initial testing, these are dimensioned just large enough to handle the Arduino's own current

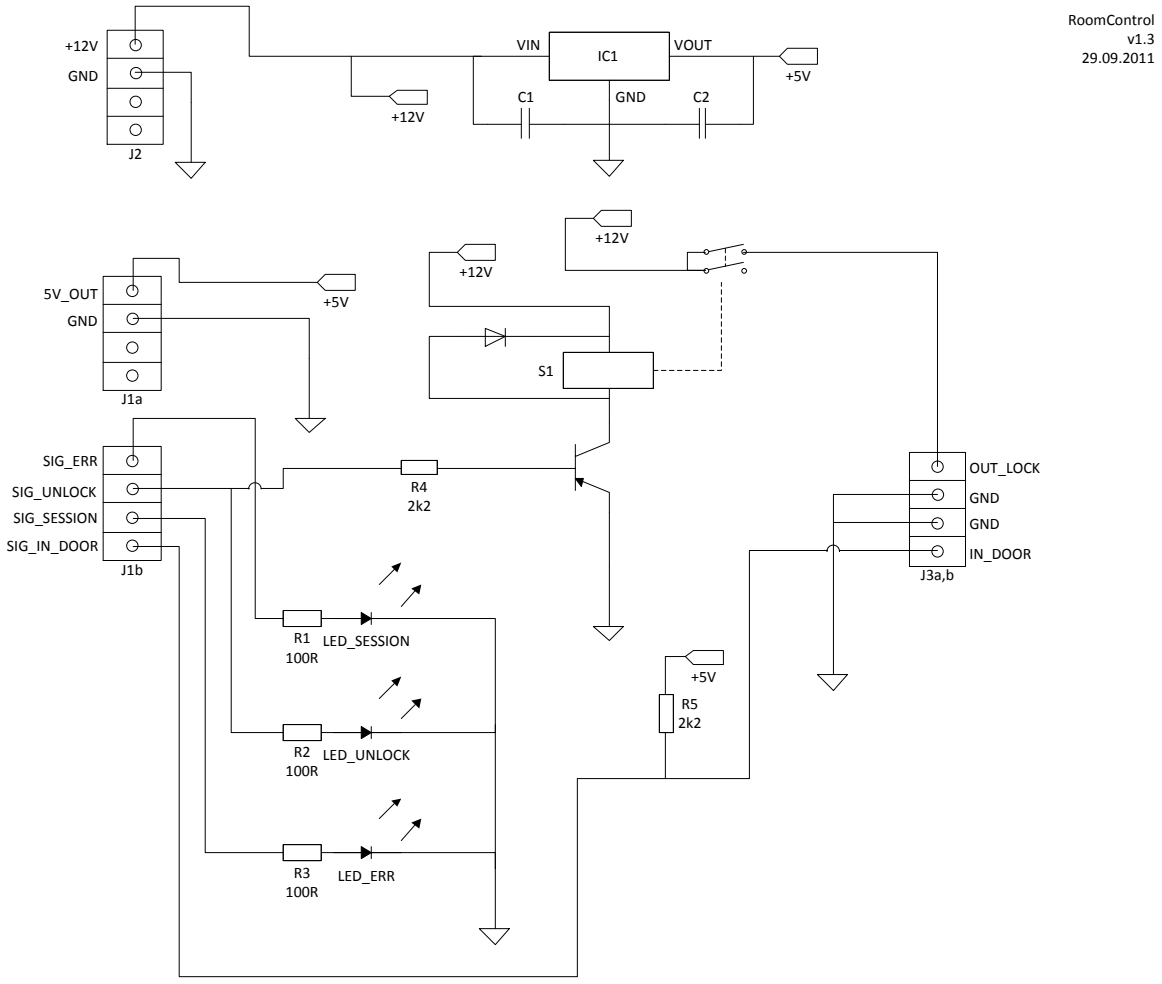

Figure 6.4: RoomControl circuit board schematic

Item	Voltage	Current
Arduino	5V - 20V	max. 30 mA
Ethernet shield	5V	max. 200 mA
Circuit board	5V	max. 180mA
Door sensor	5V	10mA
Lock actuator	12V	max. 700mA

Table 6.1: RoomControl power supply requirements

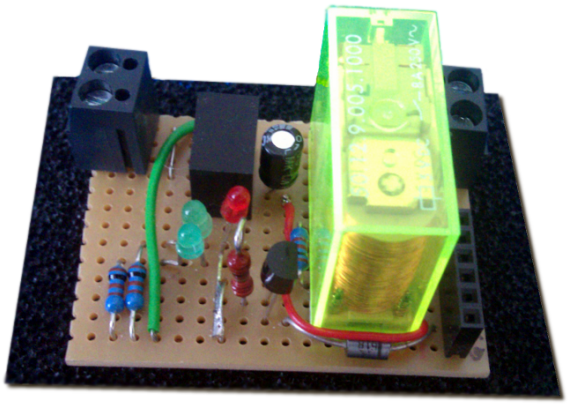

Figure 6.5: RoomControl prototype 1 circuit board

demands. Running only the Arduino and the Ethernet shield (see 6.3.7) off the internal 5V regulator (a standard linear voltage regulator) caused it to overheat and go into thermal shutdown. Clearly this is a design flaw in the Ethernet shield as its current drain can only be handled when the Arduino is running off an external regulator or USB (which provides 5V at 500mA max.). Because of this reason the internal 5V regulator is not used and the 5V supply is injected via pin 3 on the Arduino "POWER" header from the OUT_5V output on the RoomControl circuit board.

Because both 12V and 5V supplies are needed, a commercially available regulated PSU was used (see parts list in appendix B) to obtain a safe, well-regulated 12V supply from the 220V mains supply. A switching supply was chosen instead of a transformer-based design due to efficiency reasons. The PSU supplies 12V Direct Current (DC) at a maximum of 2A, which should more than suffice. The 5V supply for the Arduino, the Ethernet shield, the sensor and the internals of RoomControl circuit board (T1, LEDs, S1) is obtained from the 12V supply via the 5V voltage regulator IC1. From measurements and datasheets it was determined that about 420mA is drained from the 5V supply, so in the first prototype a standard linear voltage regulator (LM7805) was chosen for IC1. The LM7805 datasheet lists a maximum safe operating current of 1A, more than double the required 420mA. While doing the first circuit bring-up, however, it quickly became apparent that circuit cannot operate productively with this configuration. The power supply broke down intermittently (no voltage present) and the IC1 became very hot. Investigation showed that the power dissipated as thermal losses by the IC1 equate to the following,

$$P_{therm} = I * V_{drop} = 420mA * (12V - 5V) = 400mA * 7V = 2.94W$$

where V_{drop} is the voltage drop across the regulator input/output pins and I is the current through the regulator (nearly the same on the both the input and output terminals, nearly no current through the ground pin). The following calculation is the theoretical junction

temperature T_{j0} if the regulator is used without a heatsink. The thermal resistance from junction to ambient air (no heatsink used) of $R_{j0} = 65W/K$ is given in the LM7805 datasheet. With this factor the temperature difference from the junction to the ambient air temperature amounts to,

$$\Delta T = P_{therm} * R_{j0} = 2.94W * 65K/W = 191.1K$$

At $T_{ambient} = 25°C$ the theoretical junction temperature is

$$T_{junc} = T_{ambient} + \Delta T = 216.1°C$$

Clearly this is outside of the possible operating range of the package, the internal thermal shutdown circuit would regularly disrupt continuous operation. The use of a heatsink would cause a better thermal resistance to air R_{ja}, namely

$$R_{ja} = R_{jc} + R_{cs} + R_{sa}$$

where R_{jc}, R_{cs} and R_{sa} are the thermal resistances from junction to case, case to heatsink and heatsink to air respectively. Commonly known typical values for the former two are $R_{jc} = 5K/W$ and $R_{cs} = 1K/W$, assuming a standard TO-220 package and the use of heat-conducting paste to mount the heatsink. Therefore a suitable heatsink needs to be dimensioned with a thermal resistance better (lower) than the value R_{sa} determined as follows. For a safe junction temperature $T_{junc} = 70°C$ at a constant ambient room temperature of $T_{ambient} = 25°C$,

$$\Delta T = T_{junc} - T_{ambient} = 70°C - 25°C = 45K$$

is required. To ensure this, enough heat for the ΔT needs to be dissipated from junction to air,

$$R_{ja} = \frac{\Delta T}{P_{therm}}$$

Therefore, the desired heatsink should have at most

$$R_{sa} = R_{ja} - R_{jc} - R_{cs} = \frac{\Delta T}{P_{therm}} - R_{jc} - R_{cs} = \frac{45K}{2.94W} - 5K/W - 1K/W = 9.31K/W$$

Looking at possible heatsinks to fulfill this requirement, as well as the consideration that the requirement is only valid for as long as the ambient air temperature around the heatsink remains at 25°C, i.e. the presence of convection ventilation or an active fan, it was decided to search for an alternative means of building a stable 5V power supply. The LM7805 proved to be too inefficient and difficult to be run in the given conditions, especially where the RoomControl needs to be available without interruption and in unattended (and possibly adverse) environments. The alternative implemented is a much more expensive but very efficient and robust inductive DC-DC converter. The input voltage is used to transfer energy to an inductor at a regulated duty-cycle and this energy is removed from

the inductor on the output side, at the desired voltage. In contrast to the linear voltage regulation used the input and output currents are not equal, resulting in a higher transfered power and low (thermal) losses. The component used in the final RoomControl prototype has an efficiency of 94% (see datasheet supplied with this thesis' practical part's files) and needs no heatsink or ventilation up to an output current of 1A.

6.3.4 Status LEDs

Four status Light Emitting Diode (LED)s are available on the RoomControl module, either on the circuit board itself (prototype 1) or the via a terminal block for external case-mounted LEDs (prototype 2). Table 6.2 lists the indicators, their color and the state they describe.

LED Label	Color	State	Description
LED_PWR	green	lit	12V and 5V power supply is ok
LED_SSN	green	lit	A session is active (not necessarily authenticated)
LED_LOCK	green	lit	Lock actuator is unlocking door
LED_ERR	red	single blink	Power-On Self Test (POST) phase during startup
LED_ERR	red	3 blinks	Authentication failure for session
LED_ERR	red	lit	Fatal error or remote kill command (unit is offline)

Table 6.2: RoomControl LED status codes

6.3.5 Output channel

The Arduino platform has 14 digital outputs, 6 of which can provide a Pulse Width Modulation (PWM) signal, at 5V and max. 25mA, taken directly from the ATmega microprocessor chip. Clearly these outputs are insufficient to drive any but the most basic of actuators. The door lock actuator, for example, takes 600mA at 12V. Because of this reason the RoomControl controller board acts as an IO adapter for the digital outputs (see schematic 6.4). The main output channel, OUT_LOCK, is driven by a relay S1 switching the IN_12V directly. Control of the relay, in turn, is via a driver transistor circuit (R4, T1 and suppressor diode D1). Two other output channels are currently used for LED status indicators, they are driven directly through bias resistors R1 and R3.

6.3.6 Input channel

Sensors are connected to the RoomControl controller board where the necessary bias and/or de-coupling is performed. On the Arduino platform the analog inputs A1 through A6 are

used for measurement. Analog input A0 may not be connected and needs to remain floating as it is used to collect analog noise for generating non-pseudo random numbers, see section 6.5. For the door sensor the magnetic reed relay attached to the door frame (see 6.6.2) is connected to A1, and a pull-up resistor to the 5V supply. When the door state is requested the voltage is compared and a decision is made halfway at 2,5 V. In future applications, two digital inputs support continuous monitoring via hardware interrupts, however this has not been implemented in the current RoomControl software as no "alarm" function or similar was needed.

6.3.7 Network connectivity

Network connectivity is achieved by attaching an Arduino Ethernet expansion module (a so-called "shield") to the main board. The Ethernet shield is based on the Wiznet W5100 Ethernet driver chip. It interfaces to the Arduino base board via the Serial Peripheral Interface (SPI) bus and has a RJ-45 Ethernet connector. A software library is available for the Arduino IDE to assist with writing and compiling code that interfaces with the Ethernet shield. Both TCP and Universal Datagram Protocol (UDP) are supported, in both client and server roles.

6.4 RoomControl Software

Please see figure 6.6 for a high-level flow-chart overview of the RoomControl embedded software. The different states and routines are explained below.

6.4.1 Initialization

On the left side of figure 6.6 the initialization routines executed during power-up are shown. At the beginning of this process (POST) the red ERROR LED is turned on to indicate malfunction in case the boot-up code hangs during initialization. After successfully setting everything up the red ERROR LED is turned off again, this should happen after about one second. Next, the main message loop is entered, which is normally never left unless tampering with the device is detected or a remote kill-switch ("self-destruct") command is received. (In this case, the dieError() function is called which causes the red ERROR LED to remain lit and the device to be disabled by sinking execution in an endless loop.) The main message loop waits for an incoming connection from a client on TCP port 5551 and can, due to the absence of multithreading, handle only one session at a time.

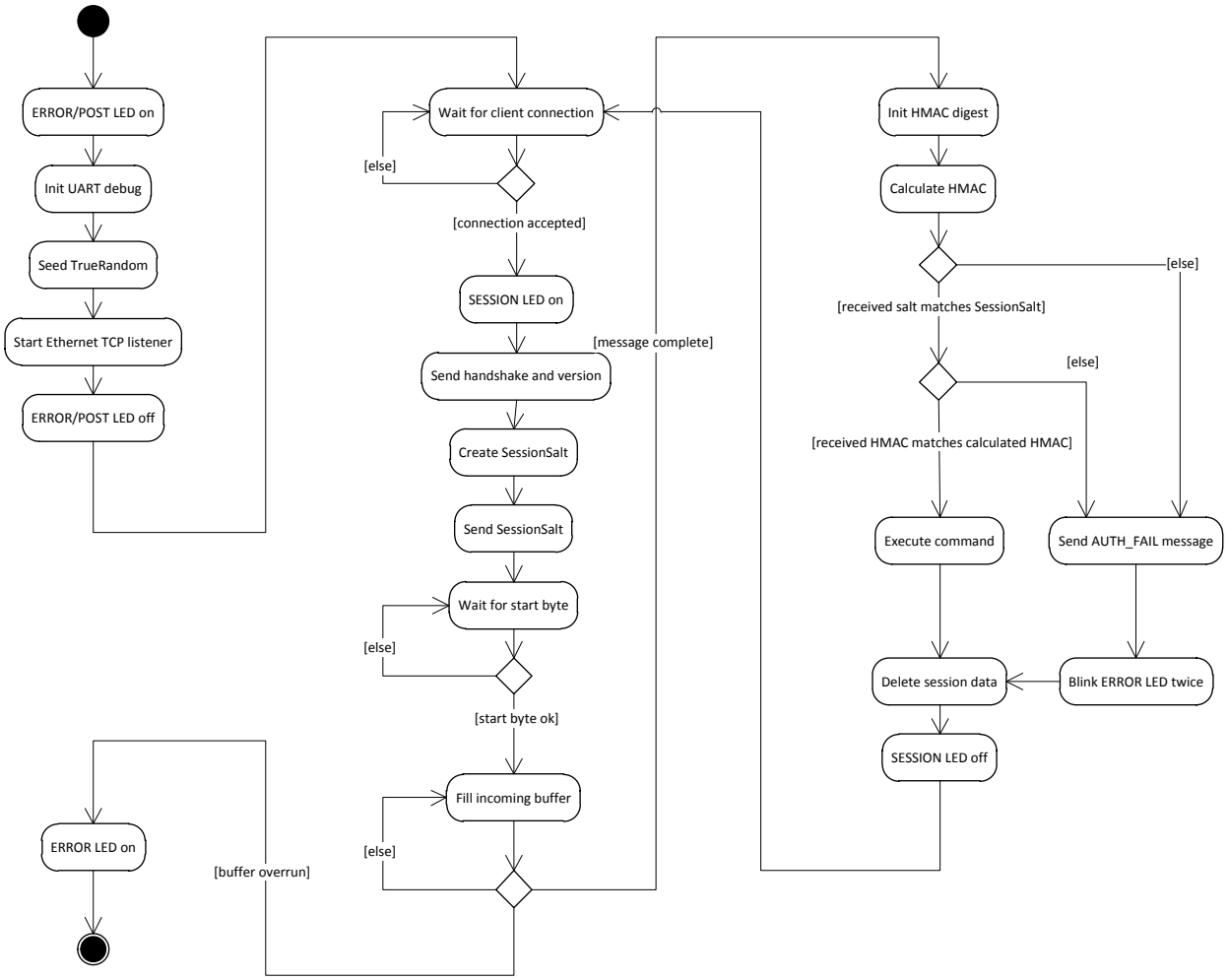

Figure 6.6: Flowchart of RoomControl embedded software

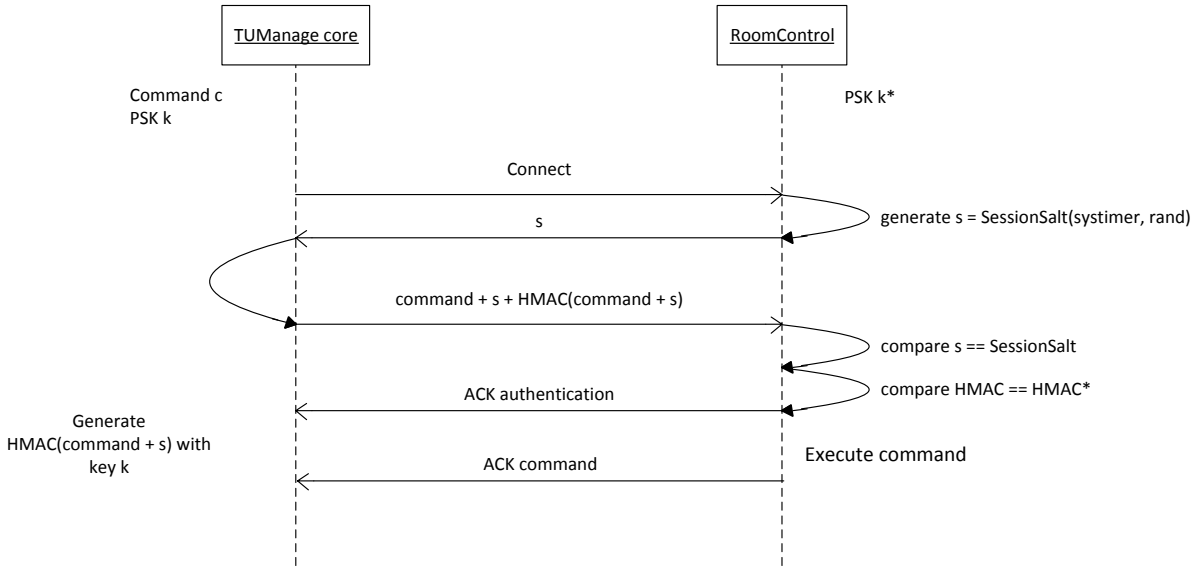

Figure 6.7: Sequence diagram for challenge-response protocol using a nonce

6.4.2 Connection handshake

Upon successful connection the SESSION LED is lit and the RoomControl's identifier version string is sent as a handshake. A 62-byte long SessionSalt number is generated where the first 4 bytes are the current system uptime in milliseconds, the remaining 58 bytes are real, non-pseudo random numbers. (Please see the section 6.5 on security below for more info on the random number generator.) The SessionSalt is sent to the client which must send the exact bytes back when sending its command. For security reasons the SessionSalt is valid for exactly one command and may never be re-used. Now the routine will wait for the start byte from the client, the American Standard Code for Information Interchange (ASCII) encoded letter 'c', and collect the entire fixed-length command message sent by the client, byte by byte, before processing it.

Start byte index	Field
0	Command (see Appendix D)
1	SessionSalt as received in handshake
63	Client HMAC
83	Terminator (one byte, either 0 or ASCII NL)

Table 6.3: RoomControl message format

6.4.3 Command Message

All command messages have a fixed length of 84 bytes. Message authenticity and authorization is performed using two techniques, salting and the use of a keyed-Hash Message Authentication Code (HMAC). For a more detailed discussion on them please see section 6.5 on security. The message structure is shown in table 6.3. Before processing an incoming message, as is depicted on the right side of figure 6.6, the HMAC digest is initialized with the secret PSK present in both the RoomControl software (the global byte-array called msgkey) and any authorized client (like the TUManage backend service). The HMAC of the Command and SessionSalt parts of the command message is calculated using the PSK and compared to the HMAC sent by the client. Also, the SessionSalt received in the message is compared to the current session's SessionSalt, as computed during the handshake phase. Should either the HMACs mismatch or the SessionSalts not be equal an AUTH_ERROR code is sent, the ERROR LED will blink twice and the session aborted plus discarded. If the two criteria are met the command is considered valid and will be processed as follows by invoking the hardware-interface layer of the Arduino platform.

In the current implementation of the RoomControl the following functions are available: Ping, get room state, unlock door for single entry, keep door unlocked until cancel message is received and the cancel message for the previous command. Please see Appendix D for a complete and detailed list of possible commands as well as their function.

6.5 Security

With the RoomControl directly connecting parts of the user's environment to a network, security and privacy are immediately major concerns. Where attackers would in the past need to be physically present to, say, break open the room's door or find out if the room is occupied, it is now theoretically possible for attacks to be performed from a remote location via the network. Possible attacks to be expected include privacy issues (when and how long are people inside the room), physical security (e.g. the door opened by an anonymous unauthorized person and items of value stolen from inside the room) as well as denial-of-service attacks (locking the door permanently to disable use of the room).

The following measures were taken to present a high barrier in terms of effort required

for potential attackers. The effort required to successfully attack the system should be considerably higher than the actual value of a successful attack. Afterwards a summary of potential weaknesses is presented in section 6.5.4.

6.5.1 Network security

A possible attack vector is via the RoomControl's Ethernet connectivity. While it is good practice to ensure that care is taken to protect the exposure of the RoomControl's software interface it is by no means sufficient to rely on this. Often it is simply not possible to completely protect the information path between the TUManage core service and the RoomControl modules or, due to the complexity of today's Internet Protocol (IP) networks, an attacker could gain access to the path by means of other vulnerable systems, e.g. by compromising a network switch or firewall. For this reason particular attention has been given to the RoomControl software security in order for it to still pose a significantly high barrier for potential attackers, even if network security has been breached.

6.5.2 Software security

The software security layer is generally considered to be the most prone to attack as it is potentially possible to gain network access to the RoomControl service when network components outside the scope of TUManage are compromised. The RoomControl software is controlled by accepting incoming TCP connections from the TUManage core service and receiving/interpreting binary command messages.

Message authenticity and integrity using HMAC

In order to be certain that the command messages are firstly unmodified during transit and secondly really originating and from the TUManage core service a HMAC is appended to every message. It is generated on both sides from the command message and a PSK which is known only to the RoomControl embedded software and authorized entities like the TUManage core service. Without the PSK it is practically impossible to generate the correct HMAC, even if numerous other message-HMAC tupels have been collected for analysis [27]. The HMAC algorithm uses two stages of hashing where in the first stage the padded key, XORed with a constant called the ipad, is prepended to the message and hashed (TUManage uses the SHA-1 hash algorithm). The output is prepended with the XOR of the padded key and another constant called the opad, whereafter it is hashed again to produce the output HMAC. The use of two stages prevents an attacker from padding a malicious input message until it matches a desired HMAC output (hash conflict). Due to this and the one-way nature of the cryptographic hash function it is practically impossible to either create or forge the HMAC of a given message without knowledge of the secret PSK

[27]. The RoomControl software therefore can reject any messages not originating from the TUManage core server or messages which have been tampered with during transit.

Replay attack prevention using a challenge-response protocol and nonce

With the message integrity and authenticity being verified using the HMAC the possibility or replay attacks, however, would still remain. It would be possible for an attacker to record all communication to and from the RoomControl until, for example, the command message to unlock the door can be captured (together with its valid HMAC). At a later time this message could be sent again to the RoomControl and it would unlock the door as the HMAC matches the command message.

This form of attack is prevented by means of a challenge-response protocol incorporating a nonce. Each command message can therefore be practically used only once and only for a limited amount of time as it is tied to the nonce. This is ensured by creating a nonce number called SessionSalt which is sent to the client during the session's handshake. Every subsequent command message must contain this SessionSalt and it becomes invalid once a message containing it has been received. No two SessionSalts may be identical and their value may not be predictable. Because of these constraints the SessionSalt is very long (62 bytes) and is generated from two values. Firstly, the system timer is used, which is an internal counter counting the number of milliseconds since the RoomControl's microprocessor has been powered up. This number is unique only for about 50 days as it wraps over to zero on the Arduino platform. The second number used is a non-pseudo random number generated from analog noise on an unused analog input pin on the Arduino. During generation of this number the TrueRandom library [28] first sets a noisy voltage on an IO pin and then measures it, discarding the all but the least significant bits of the measurement. This is repeated multiple times and de-biased using a von Neumann whitening algorithm [28]. No seeding/use of internal pseudo-random number generators is needed as speed performance is not critical while generating the SessionSalt. With the use of these practically unique and unpredictable nonces it is practically impossible to successfully perform a replay attack on the RoomControl.

6.5.3 Physical security

All components of the RoomControl need to be mounted on the inside of the door as access to the control module itself or its hookup wires could otherwise allow it to be manipulated from the outside. A simple shorting of the main PCB's power and OUT_LOCK pins could cause the door to be unlocked. Even on the inside of the room some care should be taken as to not expose the RoomControl control box too easily as a potential attacker could manipulate the internals to prevent the door from locking or cause it to wrongly unlock at a later time. This attack is, however, also possible without the RoomControl installed in a room, by simply manipulating the door itself and/or its locking mechanism.

6.5.4 Considerations for productive use

Two prominent weak points in the RoomControl security concept still remain. They would provide a starting point for future work and should be considered in productive systems. Privacy: The network communication to and from the RoomControl is not encrypted and so, if the network channel were tapped or sniffed, an unauthorized party could monitor use of the room and its door. (Please note, however, that the communication is safe from modification, as described in section 6.5.2). A good countermeasure against this attack would be to physically secure the Ethernet cable to the RoomControl and separate its network logically from other parts of the LAN, e.g. by means of network switches (which are, however, also prone to attacks such as Address Resolution Protocol (ARP) flooding) or a Virtual Private Network (VPN) tunnel if RoomControl traffic needs to traverse unsafe networks. Denial-Of-Service (DOS): The network connection of the RoomControl can be blocked if it is accessible to an attacker. As the RoomControl can handle only one session at a time, an attacker could initiate a session and never send a valid command. If the attacker then sends incorrect command message start bytes these are ignored but they would prevent the TCP inactivity timeout from terminating the connection. Sending a byte every few seconds would successfully disable the RoomControl. Countermeasures for this attack would be logical separation of the LAN the RoomClient is connected to. IP filtering alone could be circumvented as a valid source IP address could be spoofed by the attacker and pass an IP filter in one direction, but only this one direction is needed for the attack.

6.6 Actuators and sensors

6.6.1 Lock actuator with visual feedback

The easiest and most reliable method of controlling a door lock is by means of a magneto-electric strike lock. The lock is built into the door frame and when voltage is applied to the lock's terminals, the resulting current through the internal inductor causes a magnetic field to release the door pin. The door can then be pulled open. Care was taken to use a strike lock capable of continuous operation, to keep the door unlocked for prolonged periods of time without damaging the internal inductor. Also, it was required for the lock to be driven with DC by the driver circuit in order prevent the notorious buzzing sound when the door is unlocked. This buzzing sound can in some cases actually be useful to provide feedback that the door is really unlocked because this sound is widely known to the user from other scenarios. For prolonged unlocking of the door, however, the buzzing noise is annoying and so a visual feedback mechanism was chosen: A green LED on the door frame next to the door handle indicates that the door is unlocked when it is lit. In future applications a perspex door handle could be used together with internal embedded LEDs, resulting in the handle itself glowing red or green. The strike lock, being an inductive load,

has a suppressor diode fitted in reverse, parallel to the inductor, to prevent voltage spikes back to the driver circuit when current is suddenly removed.

6.6.2 Door state sensor

The state of the door (i.e. open or closed) is detected using a magnetic reed relay. The element is mounted on the door frame and the corresponding magnet on the door. If the door is closed, the proximity of the magnet pulls the reed relay to close a circuit, which is attached to the RoomControl IO board. Here, a pull-up resistor provokes the necessary voltage drop which can be measured by the Arduino's analog input. The input threshold is set in software to half the supply voltage of 5V.

6.7 Installation

The RoomControl prototype was first installed in one of TUM's lecture/seminar rooms for testing and initial usability evaluation. A card reader access control module connected to a strike lock was already installed by the building administration so it was decided to wire the RoomControl in parallel to the existing system. This means that the two unit's grounds were connected, the existing module's 12V powersupply used to supply the RoomControl and the RoomControl's OUT_LOCK output also connected to the door strike lock (in parallel to the existing system). The RoomControl box was wall-mounted next to the other system with the status LEDs clearly visible. An Ethernet cable was put into existing cable trunking and now connects the RoomControl to a nearby Ethernet switch, attaching it to a dedicated ubiquitous-computing LAN shared by other projects' components like the room's digital projectors. Network access rules for the TUManage core server to reach the RoomControl were added to the dedicated LAN's firewall.

6.8 Future considerations

6.8.1 Scaling

In future applications, multiple RoomControl modules could be used in any one room to act as sensors/actuators for their parent ubiquitous systems. Reasons for splitting the work among multiple devices could be due to logical and/or physical separation of the parts of the environment they interact with. An example for physical separation would be if a single RoomControl module were to be used to control multiple doors across the room. A single module could be used at the expense of running additional thick cables to each

Figure 6.8: Photo of RoomControl prototype installation, cover removed

door for the strike lock and status sensors, while one would on the other hand only need to run an Ethernet network feed to each door with its own RoomControl module.

6.8.2 Audit

For productive systems it could potentially be useful to provide a robust and secure means of auditing the system's use. This could be done by attaching a Secure Digital (SD)-card to the Arduino platform via one of the numerous shields available for this use. Every transaction handled by the RoomControl would then be logged to the SD-card and be untamperable even if the network communication or the cryptographic keys were compromised. Provided, of course, the embedded software is secure (low risk) and the RoomControl module is physically secure (hard to achieve once attacker has gained access to the room).

6.8.3 Robot Operating System

Related work exists ([25]) where similar devices run a standard middleware, the Robot Operating System (ROS). The currently very simple embedded software running on the RoomControl microprocessor could be replaced by a more powerful platform with ROS handling tasks such as low-level device abstraction and high-level message passing in a standard and much more scalable fashion. Compatibility with other, future work should be easier and more easily scalable if a standardized framework were used.

Chapter 7

Conclusion

7.1 Summary

This thesis identified the opportunity to assist with the efficient and convenient use of shared room resources by providing a system based on state-of-the-art research in the field of ubiquitous computing. Mechanisms for interactivity with the user were designed and implemented with special attention given to usability and security. The problem of user authentication with the system was solved using a novel approach where the user session on an unsecured terminal is temporarily moved to a secure environment (the user's personal smartphone) while the credentials are supplied to the system. Mechanisms were implemented to ensure confidentiality, two-way authenticity, integrity and non-repudiation. A scalable infrastructure was created in order to provide a robust and homogeneous experience to the user, plus enable the possibility of wide-spread deployment in a productive environment, where cost, management and maintenance overhead become important. It was proposed that the infrastructure requirements are best met using current web components plus architectures like SOA and design styles like REST. Software was designed and written for this server infrastructure and interative smart door signs based on tablet computers. Hardware was developed and built for the system to interact with the user's physical environment, using a low-cost embedded rapid-prototyping microprocessor platform. In the practical part of this thesis a complete prototype of the system was implemented in both hardware and software. It was installed in one of the lecture rooms at TUM where the system is available for evaluation and as a starting point for future work.

7.2 Outlook

7.2.1 Productive use

Where possible, the TUManage architecture was kept as open and generic as possible with many tight interfaces and a modular approach, thanks to a SOA architecture and the use

of design patterns like MVC and layers, for example. This should make additions and enhancements to the system possible without complicated workarounds and compromises. Before the system is ready to be put to productive use, however, certain aspects still need to be addressed, like for example,

Integration

The integration with existing systems like TUMonline and TUM LDAP SSO are not using productive interfaces in our prototype implementation. Care needs to be taken to ensure robust operation of these interfaces.

Administration

While this thesis placed a lot of emphasis on the usability of the frontend interfaces the backend administration interfaces are still very simple and sparse. Once non-technical room administrators need to operate and administer the system an intuitive and more powerful backend interface would need to be implemented, taking into account the exact needs of room administrators. Possible features requested could be integration with existing, private room management systems or more advanced user management with fine-grained permissions control over reservations and room use.

Hardware infrastructure

While the server infrastructure used for the prototype implementation is by far sufficient for prototype use it will probably not meet the expectations of a large-scale productive system. Administrative overhead is required for keeping the server environment running and secure. It is therefore advised to move the server infrastructure to a hosting solution where dedicated personnel ensure hardware availability and server operating system security, like for example the TUM's IT service department LRZ. Also, hardware maintenance overhead is required to ensure the smooth operation of the WallClient and RoomControl devices.

7.2.2 Multiplication

While TUManage was conceptualized to manage shared rooms, it quickly became apparent during the design phase that the system could be used to manage many other kinds of shared resources as well, notably physical objects. Care was taken to keep many components of the system as generic as possible (which is a good design trait in itself) in order to potentially adapt or port the system for application in other scenarios, which could be the basis for future work.

Appendix A

Core backend service resources

This appendix gives a list of some reseources offering service on the TUManage core backend. For the complete list please consult the documentation and the web.xml file provided in the practical part of this thesis.

DbConnection.resource	
Description:	Used to check availability of a connection to the DBMS.
URI:	`https://tueivmi-service.lmt.ei.tum.de/TUManage/admin/DbConnection.resource`
GET Method	
— Parameters:	none
— Return code:	200 OK if the DBMS connection is present, else 500 INTERNAL_SERVER_ERROR
— Returned data:	Human-readable success or error message.
— MIME type:	text/plain

SyncRooms.do	
Description:	Trigger synchronization of room data with TUMonline.
URI:	`https://tueivmi-service.lmt.ei.tum.de/TUManage/admin/SyncRooms.do`
GET Method	
— Parameters:	none
— Return code:	200 OK
— Returned data:	Basic admin UI to assemble a POST request.
— MIME type:	application/xhtml+xml
POST Method	
— Parameters:	none
— Return code:	200 OK on success
— Returned data:	Human-readable log of transaction.
— MIME type:	text/plain

SyncRoomEvents.do	
Description:	Trigger synchronization of room event data with TUMonline.
URI:	`https://tueivmi-service.lmt.ei.tum.de/TUManage/admin/SyncRoomEvents.do`
GET Method	
— Parameters:	none
— Return code:	200 OK
— Returned data:	Basic admin UI to assemble a POST request.
— MIME type:	application/xhtml+xml
POST Method	
— Parameters:	sapRoomID (ID of the room to sync), fromDate (date from when to sync, in format YYYYMMDD), untilDate (date until when to sync, inclusive)
— Return code:	200 OK on success
— Returned data:	Human-readable log of transaction.
— MIME type:	text/plain

Table A.1: Sample of administrative services/resources

Room.resource	
Description:	GET or POST information about a room.
URI:	`https://tueivmi-service.lmt.ei.tum.de/TUManage/resources/Room.resource`
GET Method	
— Parameters:	roomSapID (the room's unique ID)
— Return code:	200 OK
— Returned data:	JSON data structure containing information about the room itself.
— MIME type:	text/json
RoomEvent.resource	
Description:	GET a room's event list or POST a new event (reservation)
URI:	`https://tueivmi-service.lmt.ei.tum.de/TUManage/resources/RoomEvent.resource`
GET Method	
— Parameters:	roomSapID (the room's unique ID), fromTime and toTime (from which timespan should events be returned)
— Return code:	200 OK
— Returned data:	JSON data structure containing a list of events including event details.
— MIME type:	text/json
POST Method	
— Parameters:	see code documentation as there are numerous (optional) parameters
— Return code:	200 OK on success, 500 INTERNAL_SERVER_ERROR on failure, 403 FORBIDDEN if such a reservation may not be placed
— Returned data:	JSON data structure containing newly created event or error message of failure
— MIME type:	text/json
RoomLock.resource	
Description:	URI representing the room's lock to GET it's state or POST to unlock/lock the room.
URI:	`https://tueivmi-service.lmt.ei.tum.de/TUManage/resources/RoomEvent.resource`
GET Method	
— Parameters:	roomSapID (the room's unique ID)
— Return code:	200 OK
— Returned data:	JSON data structure containing the current state of the door and lock, including the current timestamp.
— MIME type:	text/json
POST Method	
— Parameters:	roomSapID (the room's unique ID), openMode ("single" to open the door for one person to enter, "prolonged" to keep the door open for prolongedTime, which specified in seconds)
— Return code:	200 OK on success, 500 INTERNAL_SERVER_ERROR on failure, 403 FORBIDDEN if the current user may not unlock the door
— Returned data:	JSON data structure containing the current state of the door and lock, including the current timestamp.
— MIME type:	text/json

Table A.2: Sample of room management resources

Reservation.client.resource	
Description:	Serves an instance of the HTML5/JavaScript room reservation client.
URI:	`https://tueivmi-service.lmt.ei.tum.de/TUManage/resources/WallClient/Reservation.client.resource`
GET Method	
— Parameters:	none
— Return code:	200 OK
— Returned data:	In instance of the reservation client
— MIME type:	application/xhtml+xml

Table A.3: Sample of WallClient code-on-demand resource

Appendix B

RoomControl BOM

The following table B.1 lists the BOM required to assemble a RoomControl device. Prices listed are the costs incurred while building prototype 1 and 2. Some priced are listed as zero because the components have negligible lost and parts could be salvaged from our lab's junk box.

Parent	Component ID	Component	Qty	Source	Part Number	Price	Total
Hookup (power)	PSU1	Power Supply, 12V regulated, 2Amp	1	RS	678-3925	14,99 €	14,99 €
Hookup (power)	-	Mains cable for power supply	1	Reichelt	ADC 120 SW	1,05 €	1,05 €
Mounting	-	Casing for PCB, 134x129x61mm	1	Reichelt	TEKO 022	9,25 €	9,25 €
LockControlPCB	-	PCB prototype board - 2,54mm	0,25	RS	206-5841	3,56 €	0,89 €
LockControlPCB	T1	2N2222 transistor	1	RS	544-9624	0,16 €	0,16 €
LockControlPCB	S1	Relay, DPDT, 5V	1	RS	102-490	9,69 €	9,69 €
LockControlPCB	D1	Diode	1				0,00 €
LockControlPCB	LED1, LED2	LED, green, 3mm	2				0,00 €
LockControlPCB	LED3	LED, red, 3mm	1				0,00 €
LockControlPCB	R1,R2,R3	Resistor 100R	3				0,00 €
LockControlPCB	R4	Resistor 2k2	1				0,00 €
LockControlPCB	IC1	Voltage Regulator, high-efficiency, inductive, 5V	1	RS	672-7124	6,99 €	6,99 €
LockControlPCB	J1,J2	Terminal Header, 2-pole	2				0,00 €
LockControlPCB	J3	Pin Header Socket, 6-pole	1				0,00 €
LockControlPCB	C1	Capacitor, electrolyte, 100uF, 10V	1	RS			0,00 €
Hookup	-	Ribbon cable, 6-pole	1				0,00 €
Hookup	-	Pin Header, solderable, 6-pole	2				0,00 €
Hookup	-	Pin Header, solderable, 8-pole	1				0,00 €
Hookup	-	Hookup cable for lock	1				0,00 €
Optional:							
LockControlPCB	LED1,LED2	LED, green, case mount	2	Reichelt	LED 103 A GN		
	LED3	LED, red, case mount	1	Reichelt	LED 103 A RT		

Total BOM	43,02 €

Figure B.1: RoomControl BOM

Appendix C

Arduino UNO schematics

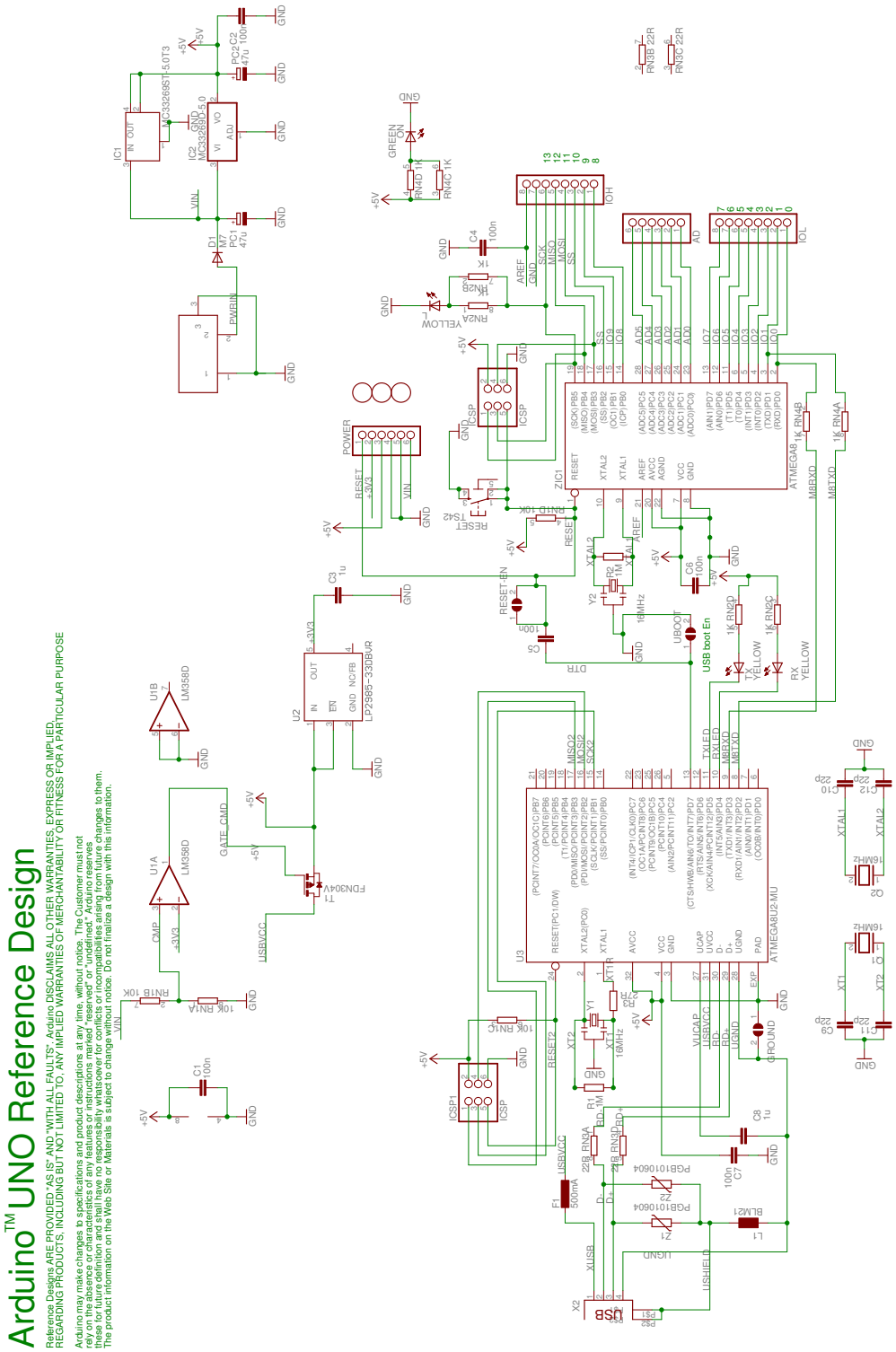

Figure C.1: Arduino UNO schematic (Source: arduino.cc)

Appendix D

RoomControl message commands

The following is a list of commands supported by the RoomControl device. Please see Chapter 6 for more details. All commands will return at least one response, on a newline terminated line, but all commands will eventually return the response OK or ERROR to indicate the command has finished processing, and with what status.

CMD_TEST	
Message byte:	t
Description:	A simple command to test and ensure communication with the embedded device is functioning.
Parameters:	None
Hardware action:	None
Responses:	The default OK line

CMD_OPEN	
Message byte:	o
Description:	Used to unlock the door for a single entry. Depending on the strike lock mechanism in use, the striker will be energized for 3 seconds in order to allow the user to pull the door open.
Parameters:	None
Hardware action:	Sets the OUT_LOCK output to HIGH for 3 seconds for the door to be opened.
Responses:	First the response BZZZZZZ is sent to indicate that the door is being unlocked. Once the entry transaction has completed the usual OK is sent.

CMD_PROLONGED_UNLOCK	
Message byte:	p
Description:	Puts the device into the prolonged-unlock state which will keep the door unlocked until the command to cancel the prolonged-unlock is received. During this phase CMD_OPEN commands are ignored.
Parameters:	None
Hardware action:	Sets the OUT_LOCK output to HIGH.
Responses:	The response for this command is PROLONGED_UNLOCK_ACTIVE and then OK as an acknowledgement.

CMD_PROLONGED_UNLOCK_CANCEL	
Message byte:	q
Description:	Cancels the prolonged-unlock state, otherwise the command is ignored.
Parameters:	None
Hardware action:	Sets the OUT_LOCK output to LOW.
Responses:	The response for this command is PROLONGED_UNLOCK_INACTIVE followed by OK.

CMD_STATE	
Message byte:	s
Description:	This command is used to request the state of the door to be served.
Parameters:	None
Hardware action:	If applicable, the state of the door will be sensed via the analog input IN_DOORSTATE.
Responses:	Two lines will be returned, the first one either OPEN or CLOSED, according to the state of the door, and the second line either PROLONGED_UNLOCK_ACTIVE or PROLONGED_UNLOCK_INACTIVE, according to the state of the lock.

Table D.1: Commands available on the RoomControl embedded device

Appendix E

Abbreviations

TUM Technische Universität München

PC Personal Computer

WSN Wireless Sensor Network

AJAX Asynchronous JavaScript and XML

REST Representational State Transfer

API Application Program Interface

UI User Interface

DC Direct Current

LED Light Emitting Diode

SPI Serial Peripheral Interface

IDE Integrated Development Environment

ROS Robot Operating System

SD Secure Digital

TCP Transmission Control Protocol

UDP Universal Datagram Protocol

PSK Pre-shared Key

ASCII American Standard Code for Information Interchange

HMAC Keyed-Hash Message Authentication Code

POST Power-On Self Test

PCB Printed Circuit Board

HMAC keyed-Hash Message Authentication Code

DOS Denial-Of-Service

USB Universal Serial Bus

PSU Power Supply Unit

BOM bill of materials

PWM Pulse Width Modulation

GUI Graphical User Interface

IO Input/Output

IP Internet Protocol

nonce number only used once

LAN Local Area Network

VPN Virtual Private Network

ARP Address Resolution Protocol

REST Representational State Transfer

URI Uniform Resource Identifier

JSON JavaScript Object Notation

XML Extensible Markup Language

DBMS Database Management System

HTTP Hypertext Transfer Protocol

HTML Hypertext Markup Language

CRUD Create Read Update Delete

MVC Model-View-Controller

QR Quick Response

ORM object-relational mapping

JDBC Java Database Connectivity

SOA Service Oritented Architecture

RPC remote procedure call

SQL Structured Query Language

HTTPS Hypertext Transfer Protocol Secure

JSP Java Server Page

SSL Secure Sockets Layer

TLS Transport Layer Security

WLAN Wireless LAN

DOM Document Object Model

CSS Cascading Style Sheets

AJP Apache JServ Protocol

SSO single sign-on

MC Mobile Client

PKI public key infrastructure

URL Uniform Resource Locator

LDAP Lightweight Directory Access Protocol

CA Certificate Authority

JDO Java Data Objects

JPA Java Persistence API

List of Figures

List of Tables

Bibliography

[1] *Apache httpd web server.* http://httpd.apache.org/,

[2] *jQuery JavaScript library.* http://jquery.com/,

[3] *jQueryMobile JavaScript framework.* http://jquerymobile.com/,

[4] *JSON - JavaScript Object Notation.* http://json.org/,

[5] *myBatis ORM data mapper framework.* http://www.mybatis.org/,

[6] *MySQL Database Management System.* http://www.mysql.com/,

[7] *Spring Application Framework.* http://www.springsource.org/,

[8] *Surveymonkey online surveys.* http://www.surveymonkey.com/,

[9] *TUM corporate design specification.* http://portal.mytum.de/corporatedesign,

[10] *Usability First - HCI Design Approaches.* http://www.usabilityfirst.com/usability-methods/hci-design-approaches/,

[11] *HTTP Authentication: Basic and Digest Access Authentication.* http://www.ietf.org/rfc/rfc2617.txt, 1999

[12] *Arduino Platform.* http://arduino.cc/, 2011

[13] *Wiring platform.* http://wiring.org.co/about.html, 2011

[14] Apache Software Foundation: *Apache Tomcat.* http://tomcat.apache.org/,

[15] Chen, Peter P.: The Entity-Relationship Model: Toward a Unified View of Data. In: *ACM Transactions on Database Systems* 1 (1976), S. 9–36

[16] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures.* Irvine, California, University of California, Diss., 2000

[17] Freeman, Eric T. ; Robson, Elisabeth ; Bates, Bert ; Sierra, Kathy ; Media, O'Reilly (Hrsg.): *Head First Design Patterns.* O'Reilly Media, 2004. – ISBN 9780596007126

[18] Google: *Android At Home Framework, public preview.* http://www.google.com/events/io/2011/index-live.html, 2011

[19] Kemper, Alfons ; Eickler, Andre: *Datenbanksysteme. Eine Einführung*. Oldenbourg Verlag, 2004. – ISBN 9783486273922

[20] Krumm, John: *Ubiquitous Computing Fundamentals*. 1st. Chapman & Hall/CRC, 2009. – ISBN 1420093606, 9781420093605

[21] Kuniavsky, Mike: *Smart Things: Ubiquitous Computing User Experience Design*. Morgan Kaufmann, 2010. – ISBN 9780123748997

[22] Mattern, Friedemann ; Flörkemeier, Christian: Vom Internet der Computer zum Internet der Dinge. In: *Informatik-Spektrum* 33 (2010), 107-121. http://dx.doi.org/10.1007/s00287-010-0417-7. – ISSN 0170–6012. – 10.1007/s00287-010-0417-7

[23] Norman, Donald A.: *The Design of Everyday Things*. Basic Books, 1988. – ISBN 9780465067107

[24] Oracle: *Java Platform, Enterprise Edition*. http://www.oracle.com/technetwork/java/javaee/overview/index.html,

[25] Roalter, Luis ; Kranz, Matthias ; Möller, Andreas: *A Middleware for Intelligent Environments and the Internet of Things*. 2010

[26] Schlichter, Johann: *Distributed Applications - Verteilte Anwendungen*. 2008

[27] Swoboda, Joachim ; Spitz, Stefan ; Pramateftakis, Michael: *Kryptographie und IT-Sicherheit*. 1st. Vieweg + Teubner Verlag, 2008

[28] TinkerIt: *TrueRandom for Arduino*. http://code.google.com/p/tinkerit/wiki/TrueRandom/, 2011

[29] Weiser, Marc: *Ubiquitous Computing*. http://www.ubiq.com/ubicomp/, 1996

[30] Weiser, Mark ; Brown, John S.: The coming age of Calm Technology. (1996)

[31] York, Judy ; Pendharkar, Parag C.: Human computer interaction issues for mobile computing in a variable work context. In: *International Journal of Human-Computer Studies* 60 (2004), Nr. 5-6, 771 - 797. http://dx.doi.org/10.1016/j.ijhcs.2003.07.004. – DOI 10.1016/j.ijhcs.2003.07.004. – ISSN 1071–5819