FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Ein interaktiver Lernansatz für Netzwerkkommunikation in der Automation

MASTERARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Boris Malinowsky
Matrikelnummer 0305551

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dr.techn. Wolfgang Kastner

Wien, 23. November 2015

Boris Malinowsky	Wolfgang Kastner

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

An Interactive Learning Approach for Domotics

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Magister der Sozial- und Wirtschaftswissenschaften

in

Computer Science Management

by

Boris Malinowsky
Registration Number 0305551

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dr.techn. Wolfgang Kastner

Vienna, 23rd November, 2015

Boris Malinowsky

Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Boris Malinowsky
Wolfgang Schmälzl-Gasse 22/11
A-1020 Wien

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

Wien, 23. November 2015

Boris Malinowsky

Acknowledgements

I am grateful to Wolfgang Kastner,
head of the Automation Systems Group (TU Vienna),
for supporting the idea of a learning platform for students.

Kurzfassung

Diese Arbeit präsentiert einen interaktiven Lernansatz mittels einer software-basierten Lernplattform für den Einsatz in der Automation. Die Lernplattform ist als Ergänzung zu Einführungsvorträgen in die Automation sowie Laborarbeit gedacht und unterstützt das Konzept des Fernstudiums.

Der Ansatz richtet sich sowohl an Schüler als auch Studenten in verschiedenen Altersgruppen. Es ist daher entscheidend, das Lernmaterial altersgerecht aufzubereiten, aber auch Rücksicht zu nehmen auf Unterschiede in der Erfahrung mit dem Thema. Die beiden Zielgruppen sind insbesondere Schüler bei Aktivitäten wie "Tag der offenen Tür" und "Kinder-Uni", sowie Studenten in Vorlesungen und Übungen im jeweiligen Forschungsbereich. Eine Priorität dieser Arbeit ist das Vorantreiben eines Designs, das eine niedrige Einstiegsschwelle für neue Anwender anstrebt, um so die Anfangsmotivation aufrecht zu erhalten und eine schnelle Umsetzung von Experimenten zu erlauben.

Der didaktische Schwerpunkt liegt im Lernen der wichtigsten Konzepte von Kommunikationsprotokollen, hier am Beispiel KNX, sowie der Arbeitsweise der entsprechenden Netzwerkstacks in Sensoren und Aktuatoren. Durch die Möglichkeit der Lernplattform Laboraufbauten nachzubauen, können Sensor-/Aktuatorsetups sowie Kommunikationsnetze schrittweise eingeführt werden, und später im Labor weitere Experimente gleichermaßen realisiert werden.

Um eine realistische Lernumgebung zu schaffen, wird der Netzwerkstack nicht simuliert, sondern verwendet die gleiche Implementierung wie reale KNX Geräte. Dies vermeidet Schwierigkeiten in der Erstellung eines exakten Simulationsmodells sowie unerwünschte Abweichungen durch die Abstraktion des Modells. Im Vergleich zu anderen Laborformen erfordert der vorgeschlagene Ansatz keine Laborgeräte, keine teuren oder restriktiven Nutzungslizenzen, unterstützt eine einfache Verteilung auf Benutzerplattformen, und das Open-Source Softwaremodell bietet Anreiz für Partizipation durch Studenten.

Das Design und die Entwicklung eines Prototypen der Lernsoftware verfolgt eine plattformübergreifende Umsetzung mittels JavaFX. Dies erlaubt den Einsatz auf verschiedenen Nutzerplattformen. Es trägt außerdem dem zunehmenden Einsatz von mobilen und intelligenten Geräten wie Tablets Rechnung.

Abstract

This work proposes an interactive learning approach by means of a learning platform solely based on software for the field of domotics. The learning platform is intended to complement introductory courses to control and automation theory and laboratory work, and supports the concept of distance learning.

The described method shall apply to a wide range of students' age. Therefore, it is crucial to provide age-appropriate education material and recognize the different levels of experience with the subject matter. In particular, the two target groups are young students participating in children's university activities ("Kinder-Uni"), and graduate level university students participating in courses in the respective field. This work encourages a design solution which maintains a low entry threshold for beginning users, to uphold motivation and allow rapid prototyping for experiments.

The didactic focus is to cover the main concepts of communication protocols by example of KNX, and the functioning of network stacks in sensors and actuators. By allowing to resemble laboratory setups, the aim is to gradually introduce sensor/actuator setups and communication networks, which can later be realized for laboratory experiments.

To provide a realistic learning environment, the network stack is not simulated but uses the same implementation as in physical KNX devices. This avoids the challenge of creating an accurate simulation model and introducing unwanted divergency through abstraction. Compared to other types of laboratories, the proposed software solution improves in that it requires no laboratory hardware or per-seat licencing, supports easy deployment to user platforms, and has the incentive for student participation by following an open-source software model.

The design and development of the software prototype adheres to a cross-platform implementation in JavaFX. This accounts for different computer platforms and the increasing replacement of traditional workstations with mobile devices especially attractive to younger students.

Contents

CHAPTER 1

Introduction

1.1 Motivation & Problem Statement

Teaching in the field of distributed automation systems involves the study of control systems, embedded or System-on-Chip (SoC) hardware used for sensors and actuators, communication protocols, as well as their standards.

The last decade brought a steep increase in applying automation solutions, leveraged by emerging "smart" topics like *Smart Home* or *Smart Living*, *Smart Grid*, and *Smart Metering*. This phenomenon led to a high awareness and penetration of everyday life, and correlates with the movement towards enabling the Internet of Things (IoT). Alongside that development is the trend to improve interoperability of control systems, and build frameworks that support interaction of heterogeneous installations. The user benefit is ubiquitous access, with a wide range of applications for monitoring and control over wireless equipment, e.g., cellular phones.

The field of domotics – the automation of buildings, homes, and household appliances – is of particular interest for two reasons:

1. Automation as emerging key factor in future building architecture and infrastructure, enabling the *smart* home.
2. The "invisible" aspect, with distributed automation systems being embedded in infrastructure, in contrast to, e.g., automation in industrial plants.

The first reason shows the importance of automated solutions in an increasing number of people's life, and directly impacts living. The second reason is a motivator to introduce and emphasize the theory and underlying concepts of domotics to a broader audience.

As for change in human perception and the "invisibility" aspect, another example are the fundamental changes that modern wireless communications and smart devices like cellular phones and tablets introduced to users. For digital cellular networks this involved major upgrades, from 2^{nd} generation (2G) mobile network to current developments of 5^{th} generation (5G) telecommunication standards. Those advances shaped user communication behavior as well as expectations towards those technologies, but the required infrastructure is mostly invisible in day-to-day applications. Similar to the evolution that enabled smart devices but at a slower pace, developments in domotics shape human perception of, and interaction with, buildings and homes. And today, the ubiquity of smart devices became an important factor in studying and designing ways of human interaction with smart home solutions.

To further the concepts of domotics, students are a predestined group to benefit from ongoing topics in teaching and research. One disadvantage for students interested in that field is that often the necessary equipment cannot be directly obtained due to the high investment costs. Therefore, one is limited to laboratory resources at the university. Even in research and development (R&D) of control systems in the industry, often only a small set of devices is available for testing.

The core problem is in providing students with an alternative, a learning platform without those restrictions. Specifically, a platform for introducing the concepts of automation networks and communication protocols without the requirement of physical equipment.

The challenge is to ensure a sufficiently accurate representation of communication network and protocols. Ideally, the protocols should be based on the same implementations as used for devices in physical setups. For a learning platform to fulfill its educational purpose, the network structure, executed protocols, and abstracted equipment shall assimilate setups deployed in the field. Another challenge is to visualize the concepts of automation to students, without presuming detailed knowledge of network and protocol internals.

1.2 Relevance to the Field

Teaching methods and the different approaches to learning are an integral part of the curriculum of *Informatics Management*. Applying new methods to the field of embedded distributed systems and automation can help guiding students in one of the key contributors of future infrastructure. The significant impact shall be the availability of age-appropriate educational material, and avoidance of cost-intensive equipment.

Currently, the prerequisites on equipment limit the educational horizon to either theory-only, or students already enrolled in that particular field of research.

Providing a learning environment that also addresses students at a young age strengthens the didactic approach of the anticipated software solution.

1.3 Aim of the Work

Current approaches in studying control systems involves the theory, with literature covering overall concepts, standards, and algorithms. The practical side is concerned with laboratory tasks, prototyping protocols, and experimental setups. A third point can be the experience in field excursions, visiting deployed systems.

This work supplements those approaches in provide a learning platform solely based on software. The aim is to describe a method to integrate and use such learning platform alongside the introduction of concepts in domotics, and laboratory work of students. In addition, such platform furthers the ideas of distance learning in the topics of control and automation systems. The described method shall be applicable to a wide range of students' age, therefore, recognizing the different levels of experience with the subject matter. In particular, of interest are the following groups:

- young students approaching middle school, participating in children's university activities ("Kinder-Uni"), and
- graduate level university students participating in courses in the respective field.

One assumption here is that computers *do* change the thinking processes of human beings. Following that assumption, young people will actually more intuitively interact with a virtual learning platform dedicated to an education topic in computer science.

Existing frameworks and platforms aim for ease-of-use for the user to access *existing* automation setups. Hence, they require physical setups. In contrast, the resulting platform as described in this thesis supports the introduction to the concepts of domotics from a teaching perspective, not requiring specific equipment. The aim is to gradually introduce a setup consisting of the communication network, sensors, actuators, and controllers. The learning platform shall allow resembling laboratory setups. From a user perspective, the platform addresses the learning goals of the two target groups. For graduate students, the platform provides the necessary guidance to apply the learned material, and allows rapid prototyping and experimentation with setups. Those setups might later be combined and evaluated in laboratory experiments. One advantage for students should be the easier application of knowledge – gathered from interaction with the learning platform – to an actual physical laboratory setup. To address the didactic approach with children's university activities, the platform's graphical design should follow an age-appropriate introduction to the topic by playful interaction.

In aiming at eliminating the requirements on laboratory equipment, a similar goal with respect to the practical part of the work is reducing specific hardware and runtime requirements of the learning platform. Hence, the aim here is to specifically consider the various computer platforms of the targeted user groups, whether it be desktop computers or smart devices.

1.4 Expected Results

The expected results can be grouped into a theoretical part and a practical part. For the theoretical part, the results describe the design of a learning platform, which provides a graphical user interface to interact with building control. The user-centered design shall accommodate the purpose of introducing the main principles in distributed automation systems. Here, the results will specifically cover communication and applications in building automation. Example applications for monitoring and control are lighting, shading (shutters and blinds), Heating, Ventilating, and Air Conditioning (HVAC), energy management, and smart metering.

The practical part is a proof-of-concept implementation in software. The computer software follows a cross-platform approach to maintain a high portability of the resulting learning platform. The KNX standard [1] is used for communication. Here, a focus is to provide and visualize the required communication on network level as specified by the applied KNX automation standard. Specifically, the abstraction and virtualization of network links for particular communication media, to exemplify the differences in frame formats depending on the transmission medium. Another focus is extending the ability of programmatic data extraction for higher-level presentation, e.g., visualization in a user interface. This shall deepen a student's understanding of the used exchange formats, message sequences, and transmitted information.

For the graphical user interface (GUI), the design shall specifically target users at an introductionary level. Prerequisites common in engineering tools and laboratory software shall be kept at a minimum, e.g., network and device configuration required before use, and mandatory setup procedures, be it wizard-guided or via modal dialog sequences. Instead, the GUI shall allow an exploratory approach by the user, and be non-intrusive to the work-flow preferred by her. The prototype focus is on applying the following two design approaches: 1) symbols of stylized pictures of domotic devices, providing simple animations of sensor and actuator functionality for user feedback, and 2) controls providing *rich notifications* using popup windows, e.g., implemented based on balloon or tooltip controls.

When operating the platform in *graduate mode*, the platform shall provide configuration possibilities to accommodate for the specific task assignment. One example are tasks that involve configuring the platform to resemble a given physical setup. For that purpose, a suitable configuration format shall be used. Here, the main criteria are easy data exchange and data persistence, as well as the user's ability for direct modification. The proposed method in this thesis is to provide a text-based, structured mechanism, e.g., using markup language. Additionally, splitting the configuration into multiple files located within a directory layout that resembles the device types hierarchy in the platform should provide a clear and self-contained configuration layout.

The resulting platform design will offer two dedicated learning modes for interaction, to address the two target groups: attendants of children university activities (*junior mode*), and graduate students (*graduate mode*).

1.5 Methodology & Approach

The following list outlines the methodology and approach applied in this work

1. Start with a literature review, specifically on recent developments in learning and laboratory concepts in the field of domotics, summarized as state of the art.
2. Distill requirements for the learning platform design, distinguishing the two targeted user groups, differing in age, educational background, and interest. Derive and discuss specific design requirements for keeping a low entry barrier for users of the learning platform to maximize its didactic purpose.
3. Propose the necessary design elements for a cross-platform approach to allow distribution on all common computer platforms. Provide a survey of the different communication protocols and existing network libraries, also in correlation to licensing and programming language selection.
4. Establish the model and operation of a learning platform, featuring *a)* the communication protocols used in a distributed automation system, *b)* the sensor, actuator, and controller device types, and *c)* the communication network infrastructure.
5. Proof-of-concept, developing a software prototype based on the proposed platform design
6. Qualitative iteration by means of reviewing the software prototype; a quantitative evaluation is conducted by creating basic communication setups and executing communication procedures which shall correspond to hands-on experiments in the laboratory.

1.6 Structure of the Work

This chapter – Introduction – provides a motivation for the selected topic, a statement of the research problem and expected results, as well as an outline of the thesis with its considered methodology and approach.

Chapter 2 – Background – starts with a discussion of the state of the art for learning platforms and remote laboratories in the field of domotics. Subsequently, the chapter provides a short summary of popular, wide-spread communication protocols for control and automation. Specific sections are dedicated to KNX, the standard used for prototyping the learning platform for the practical part of this thesis, as well as an overview of open-source libraries appropriate for that task.

In chapter 3 – Design of a Learning Platform – the design of a learning platform that meets the individual needs of the target groups is introduced. The chapter provides the platform's approach for learning, experimentation, self-studying. Here, one important aspect is the GUI, discussed in section 3.3, together with selecting a suitable user interface toolkit or framework. Students should also be able to modify and enhance the learning platform during their laboratory exercises. For that reason, section 3.4 covers the aspects related to selecting an appropriate programming language.

Chapter 4 – Model and Operation – details the various models for abstracting specific behavior of the learning platform. This includes the communication model of KNX, device model, and models for sensor and actuator data. Specifically, that chapter of the thesis describes the application of an energy model and a weather model. Section 4.4 specifies the integration of the KNX network layout and the protocol stack. In addition, the section describes the approach to support external access to the learning platform via a gateway based on the KNXnet/IP specification. The Model and Operation chapter concludes with a use-case for automating an apartment that serves as input for prototyping the learning platform.

Conclusion & Outlook, chapter 5, provides a summary of the thesis' subject, with a final discussion of the contribution. It highlights the importance of an interactive software-based learning approach in domotics, together with providing an age- and knowledge-appropriate learning environment. The chapter concludes with a list of open questions for future work, conveying ideas for enhancing the specified design and prototype.

CHAPTER 2

Background

This chapter starts with a section on the state of the art of the different approaches in laboratory learning. Subsequently, the various concepts and forms of laboratory learning are categorized. This shall establish a common understanding and indicate the differences in the approach as proposed in this work. Section 2.3 provides an overview of common protocols for control and automation. A section is dedicated to KNX, the communication standard used in the learning platform. For using KNX functionality, the various available libraries are discussed, with focus on open-source and free implementations. The chapter concludes with a detailed introduction to the Calimero project, implementing the network stack selected for prototyping the learning platform.

2.1 State of the Art

Research institutes in the field of dependable distributed systems and distributed automation systems operate laboratories dedicated to control systems, e.g., the A-Lab of the University of Technology Vienna [2]. This provides students with the opportunity to experiment with up-to-date equipment in the lectured topics of science. Laboratory is also an important part in the curricula of computer engineering, completing the theoretical lecture parts by performing scientific exercises and measurements.

An early comparative literature review of hands-on, simulated and remote laboratories is provided by the authors of a survey in 2006 [3]. Most of the reviewed literature had as subject mechanical and electrical engineering. The authors discuss the differences in laboratories, either leveraging design skills versus conceptual understanding, and observe the lack of standardized criteria to judge on the educational effectiveness of laboratory work. Another observation is that most laboratories are already mediated by computers. The main conclusions by the authors are that researchers perhaps over-attribute learning success to the technologies used, and that the belief of students in laboratories might affect the effectiveness of laboratories.

An overview of the evolution and combining physical, remote, and virtual labs in the area of control and automation is given by the work in [4]. The authors also discuss embedding labs into E-Learning systems, and a three-dimensional virtual laboratory. One conclusion of that study is the importance of student collaboration, achieved by embedding new forms of educational laboratories into e-learning systems and Computer-Supported Collaborative Environments (CSCEs). Based on the authors' experience, such CSCE should include a shared media workspace, 3D social interface, feedback and content adaptation, integration with e-learning systems, tutoring systems, students collaboration, augmented senses immersion, and serious game concepts [4, Section III].

Especially the last point – game concepts – is interesting for this work on an interactive learning approach for domotics in relation with younger students. At a younger age, basic concepts of automation are used in games. One example is the field of robotics, e.g., LEGO Mindstorms [5], suggested to children from a minimum age of around 10.

With a teaching focus on embedded system solutions for graduate students, a commercial tool commonly used is Proteus Virtual System Modelling (VSM) [6]. It offers co-simulation of complete micro-controller based designs and supports virtual components and instrumentation. The extensive tool environment provides useful insight applicable to related research topics like distributed automation systems. But relying solely on such tooling is not sufficient in teaching more complex systems [7], e.g., control systems found in domotics. This is one conclusion of the authors as part of their approach of a distance learning course for embedded systems [7]. With the initial approach of handing out hardware to students, the authors recognized several drawbacks. Those stem from the overhead of distributing, collecting, and servicing the required hardware. A cost factor is the required update cycle of a few years to keep all equipment aligned with advances in embedded technologies. Because students in the distance learning phase cannot build upon real hardware anymore, Proteus was used for that purpose. As initially described, Proteus was considered not sufficient for more complex systems, leading to the development of a remote lab for distance learning course, with direct access to real laboratory equipment.

Using a fieldbus simulator developed in LabVIEW, the authors of [8] evaluate the learning process of students in a distance education setup. In their conclusion, the authors point out the positive effect of the GUI with its intentional similarity to real fieldbus configuration software. Additionally mentioned is the lower cost of simulation and less restrictions in experiments. Points for improvement are the complicated setup procedures and lack of help support.

Also related to the problem of complex setup phases is early work in [9], which discusses the difficulties for students in domotic modeling to create the intended domotic simulation environment. The authors propose an intermediate solution using a plan editor. Such planning step removes some complexity of the modeling task, helping the student in problem solving and to derive a solution. Based on that insight, the learning platform proposed in this work will use device templates to mitigate the initial planning overhead.

In advancing remote laboratories in terms of *smart device networks*, the work in [10] discusses a service-oriented infrastructure, including device discovery, scheduling and reservation, and learning analytics. Still, the overall assumption is that a certain minimum of laboratory equipment is available to serve remote access.

A common approach both at universities and in industry is to use training-equipment in the format of cases or trolleys [11]. The included installation of mounted and pre-configured network devices allows for a quick setup and execution of projected task assignments. Having an advantage due to their mobility, the disadvantage is – as with laboratory resources in general – the limited availability to a number of students. Figure 2.1 shows one example of such an installation inside a case.

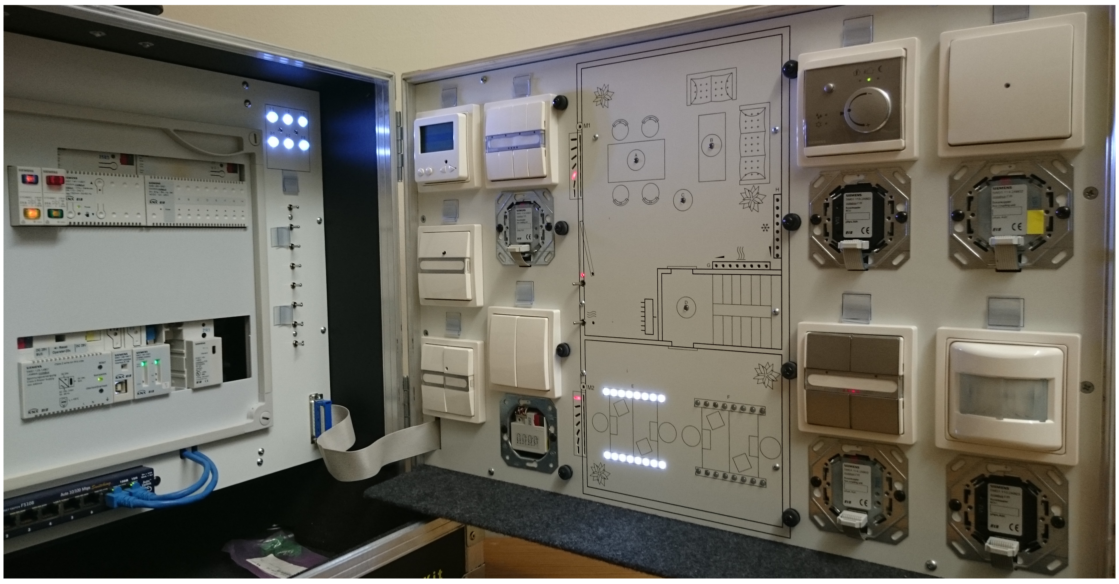


Figure 2.1: Siemens GAMMA Trainings Kit of the A-Lab.

As part of the state of the art survey, to mention is the open communication standard envisioned for use in the learning platform, ISO/IEC 14543 (KNX) [1]. The comprehensive standard specifies communication for building control and automation in the domains lighting & shading, HVAC, load management, and metering. This covers most requirements in future intelligent buildings. The standard supports several transmission media, both wired and wireless. Therefore, KNX is a good choice for a wide range of use-cases introducing communication concepts in domotics.

2.2 Concepts for Laboratory Learning

The learning process of a student is usually guided and influenced by several parts of education provided in a teaching course, e.g., at the university. The main parts include lectures, learning material, laboratory assignments, and maybe excursions. In the remainder of this section, we take a closer look on newer approaches for laboratory learning.

The traditional setup of laboratory requires the physical attendance of the students. The students either take part in guided experiments with a laboratory assistant, or are assigned tasks to solve individually or in groups applying the knowledge from lectures and learning material. In comparison to other forms of laboratory learning, this is also referred to as *hands-on labs*.

Simulated laboratories provide a user with an environment for experimentation based on simulation models. The advantage of a simulation model is that no physical setup is required. Criteria for using simulated laboratories are the availability of an appropriate simulation model, and its availability to students. Certain limitations might include high minimum hardware requirements, e.g., for computation-intensive simulations, or workstation-based licensing. Another criteria is whether a (near) real-time simulation behavior is desired and possible. One example of applying a simulated laboratory is using Modelica [12] within a development environment like Dymola [13].

One category of online laboratories is a *remote lab* setting. Here, the laboratory students are provided with the possibility to remotely control the equipment under study. The main consideration with such setting is to allow working with a physical setup, as compared to a simulation setup. At the same, this consideration also provides the main benefit for a student, by not introducing any errors, abstractions, or over-simplifications of a simulation model. Where necessary, visual feedback is commonly supported via installed cameras capturing the equipment. For very expensive equipment, remote laboratories also allow cost-sharing among several institutes or universities.

The authors of [10] distinguish between interactive and non-interactive operation. While in an interactive setup, parameters of the setup can be changed during the experiment, a non-interactive setup resembles a batched processing.

Interactive operation has certain minimum requirements on the network connection to allow a fluent workflow. Main network parameters include packet round-trip times for issuing control commands and receiving control feedback, and connection bandwidth for video streaming. With web-based remote laboratories, many communication protocols are available for the various types of communication; an overview and categorization of common protocols gives figure 2.2.

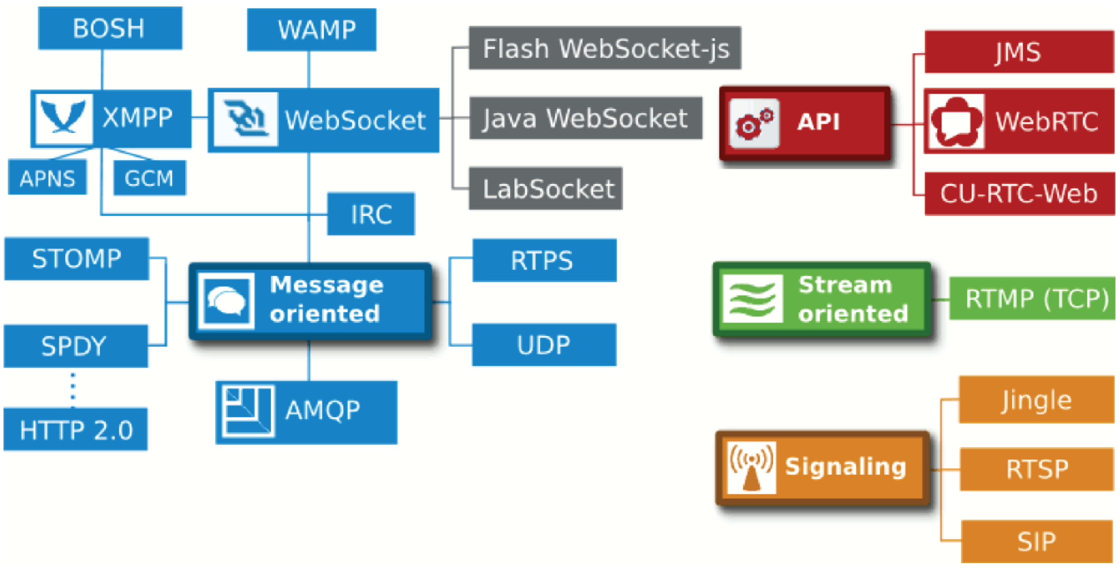


Figure 2.2: Soft real-time protocols for remote laboratories [10].

2.3 Protocols for Control & Automation

This section provides a short selection of widely recognized and deployed protocols for control and automation, with focus on Europe. Of interest are automation standards that are applied in domotics.

2.3.1 EnOcean

EnOcean protocols [14, 15] are designed for very energy-efficent transmission of telegrams. One distinctive technology feature of EnOcean is batteryless transmission and reception of wireless telegrams. Instead, this technology is based on piezoelectricity, kinetic energy, or solar cells. Hence, the main advantages are deployment in locations without electrical power supply, together with the elimination of battery charging/exchanges, but also using the protocol implementations for low-power devices.

2.3.2 KNX

For KNX, see the corresponding section 2.4.

2.3.3 ZigBee

Based on IEEE 802.15.4 [16], ZigBee specifies wireless communication protocols that mainly aim at personal area networks (PANs), to be implemented in low-cost devices. ZigBee provides dedicated specifications for use in domotics, like *home automation*, *building automation*, *smart energy*, *light link*, and more.

2.3.4 BACnet

Building Automation and Control Networks (BACnet), standardized according to ISO 16484-5 [17], is a protocol for building automation and control system networking. It is built upon an object model to represent the various device functions as standardized attributes. Each object contains properties, consisting of identifier, data type, and conformance code, relevant to BACnet services. The specification includes several communication media, with wireless transmission added as addendum which defines the use of ZigBee as BACnet data link layer [18]. BACnet also specifies a layer for data exchange using LonTalk, a LonWorks protocol (see section 2.3.5).

2.3.5 LonWorks

LonWorks (Local Operating Network) is a platform for control-networking systems, its communication protocol is LonTalk specified in ISO/IEC 14908-1 [19]. LonWorks solutions are deployed in industrial and building control systems, as well as home automation. Communication is based on Standard Network Variable Types (SNVTs), specifying an agreed-upon semantics used by all applications.

2.4 KNX Standard

KNX is standardized as international standard ISO/IEC 14543-3, as European standard CENELEC EN 50090 and CEN EN 13321-1, and as Chinese standard GB/T 20965.

The specified protocols in KNX are aligned to the Open Systems Interconnection (OSI) conceptual model [20]. On the physical layer, several communication media are supported. To better understand the reason for the various communication media, knowing the KNX heritage is useful. When KNX emerged in 2002 as successor of the European Installation Bus (EIB), BatiBUS, and European Home Systems Protocol (EHS), the physical layers Twisted Pair 0 (TP0) and Power-line 132 (PL132) were integrated from BatiBUS and EHS. The main contributor to the successor standard was EIB, having most of its specifications merged and physical layers integrated.

The communication media initially supported by KNX consisted of [21]

- TP0

- Twisted Pair 1 (TP1)
- PL132
- Power-line 110 (PL110)
- Radio Frequency (RF)[1]
- Infrared (IR)[2]

Although not directly a communication medium, KNXnet/IP specifies access to KNX networks over Internet Protocol (IP) networks.

TP0 and PL132 got classified as phased out in 2008. With the approval of RF [23] and specification of KNX communication over IP (KNX IP) [24], the current KNX communication media are

- TP1
- PL110
- RF
- KNX IP

(as well as the KNXnet/IP specification).

The KNX external message interface formats are External Message Interface 1 (EMI1), External Message Interface 2 (EMI2), and common External Message Interface (cEMI).

For process communication, data is exchanged through *datapoints*. Datapoints use standardized type definitions, the so-called Datapoint Types (DPTs), for format, encoding, range and unit. Applications in a KNX network communicate via *Functional Blocks* [25], which exchange information on the network using one or more *datapoints*. Therefore, a functional block in KNX describes a specific task of an application. From a communication point of view, KNX distinguishes four classes of datapoints [25]:

1. Group Object Datapoint
2. Interface Object Property Datapoint
3. Polling Value Datapoint

1 Specified, but no KNX approved standard document.
2 Data link layer was not specified [22, section 1.2].

4. Memory Mapped Datapoint

From an *access* point of view, datapoints can be categorized as [25]:

1. Input: a value received and processed by a functional block
2. Output: a value result of a functional block
3. Parameter: controls the function f for evaluating Output $= f(\text{Input})$, usually set by management functions
4. Diagnostic Data: local or internal status information

KNX supports two main modes of device configuration:

a. System installation mode (S-mode) requires planning of the installation and configuration using the Engineering Tool Software (ETS). The ETS is a manufacturer independent configuration tool distributed by the KNX Association [26].

b. Easy installation mode (E-mode) allows configuration using a central controller or using push buttons on the installed devices. It does not require the ETS.

The overall KNX standard is quite comprehensive; for further details, the interested reader is referred to the references supplied in this section, as well as to technical literature, e.g., see [27].

2.5 KNX Libraries

With the learning platform intended for pupils and students, and to be used for non-commercial, academic and teaching purposes, the learning platform software should be free to use. First, no costs should arise by depending on software components having license models limited by number of licensed workstations, per-seat licenses, or time restrictions. Second, software components should be available as open-source and permit software modifications and distribution.

Two main criteria for considering a KNX library are therefore *a)* free use, and *b)* availability under a common open-source license.

The remainder of this section provides a non-exhaustive overview of libraries. Libraries are chosen to cover different programming languages, providing fundamental features for at least one way of KNX communication. Libraries that act as wrapper to other KNX software libraries are not included.

2.5.1 KNX.net

KNX.net [28] is a C# implementation of a KNX Application Programming Interface (API) for KNXnet/IP tunneling and routing. Its main use-case is KNX process communication. The API supports a small set of common DPTS[1]. It is distributed under the MIT license.

2.5.2 BCU SDK

The Bus Coupling Unit (BCU) software development kit (SDK) [29] is a free development environment implemented using the C language for BCU types 1 and 2 in EIB. A main component of the SDK is eibd (EIB *daemon*, a common type of abbreviation in Linux) [29, chapter 7], providing several interfaces for KNX communication. As of the latest version 0.0.5[2], eibd supports Physical External Interface (PEI) 16, FT1.2, EIBnet/IP Routing & Tunneling, as well as Twisted-Pair Universal Asynchronous Receiver Transmitter (TP-UART). The big set of supported features non-withstanding, eibd does not support the translation of DPTS.

SDK distributions are available as source archives, RPM Package Manager (RPM) and Debian packages. Most work is licensed under the GNU General Public License (GPL), certain parts use the Lesser GNU General Public License (LGPL) or Berkeley Software Distribution (BSD) license.

2.5.3 knxd

knxd [30] is a fork of eibd (see section 2.5.2), initially driven by the wish to integrate certain bug fixes and address problems in the interaction with ETS 5 [31].

2.5.4 pKNyX

pKNyX [32] is a Python KNX Framework, aiming at KNX process communication. The initial target of pKNyX was to provide a pure Python implementation, without the need for cross-compilation of KNX logic written in other languages. A main design aspect of the library focuses on *functional blocks*, providing the logic for datapoints exposed by devices. Part of the source code was ported to Python from Calimero (section 2.6). pKNyX is released under the GPL.

2.5.5 Calimero

A Java-based implementation; see section 2.6, which provides a detailed introduction to the Calimero project.

1 At version 1.1.3, 9 DPTS are supported, for DPTS with main numbers 3, 5, 6, and 9.

2 The version numbering is not representative for maturity and stability; most components are in stable use for years.

2.6 The Calimero Project

The Calimero Project is dedicated to provide Java libraries for KNX access and management. The name Calimero origins from an inside joke[1]: Calimero is not a falcon, referring to the Windows Distributed Component Object Model (DCOM) KNX Falcon driver library. With the project started in 2005, initial library versions were available via the `eicl` packages for EIBnet/IP tunneling, translation of common DPTs, and process communication [33]. From version 2.0 on, libraries were added for KNXnet/IP server-side functionality and KNX device communication. Current development is hosted on GitHub [34]. With maven, Calimero uses the group identifier `com.github.calimero`. Bundled versions are also downloadable from SourceForge [35]. Later Calimero versions use the GPL with the Classpath Exception[2] [36].

2.6.1 Runtime Requirements

The initial development and the distributed eicl packages targeted Java Standard Edition 1.4. With Calimero NG[3], one major change was the execution on embedded system platforms. The minimum required target environment became Java Micro Edition (ME) Connected Device Configuration (CDC) with the Foundation Profile (FP). With some minor exceptions – mainly `javax.microedition.io.*` – the Java ME API is upwards compatible to Java Standard Edition (SE). Hence, Calimero always supported Java SE as runtime environment.

The advent of Java 8 brought compact profiles to Java, that enable a reduced memory footprint. The specified three profiles are strict subsets of the Java SE Platform API, forming the transitive dependency $compact1 \subset compact2 \subset compact3 \subset$ full SE API [37]. Therefore, Java 8 specifically targets again execution in resource constrained environments.

Calimero requires the smallest profile, *compact1*. Java SE Embedded 8 with the compact1 profile is similar to Java ME CDC with the FP, providing an environment for headless applications on embedded devices that require a small footprint.

On a practical note, future library versions, starting with version 2.3, have a minimum required runtime environment of Java SE Embedded 8 with the compact1 profile [37]. The software prototype developed as part of this thesis has as minimum requirement Java 8 (see also appendix A on page 52).

1 Private communication.
2 The license was changed from GPL shortly before version 2.1.
3 For "Next Generation".

2.6.2 Features

The Calimero libraries provides support for various tasks regarding KNX, covering different communication protocols, communication media, translation capabilities of datapoint types, and management tasks. For a categorization with respect to application domain, Calimero distinguishes between core functionality, server, and KNX device functionality. This is also reflected in the source code repositories for development as well as in the distribution via `jar` archives.

The remainder of this section provides an overview of the main features.

Communication Media

Calimero supports all of the current communication media specified by KNX, i.e., that are not phased out; see section 2.4 on page 12 for the list. It provides access using the appropriate communication protocols on link layer and above, together with the required External Message Interfaces (EMIs).

Access Protocols

The following access protocols are supported for a client or server to access a KNX network:

- KNXnet/IP
- KNX IP
- KNX RF Universal Serial Bus (USB)
- KNX USB
- KNX FT1.2 Protocol (serial connections)
- TP-UART for the TP1 communication medium

KNXnet/IP This protocol is implemented with the following support:

- Discovery and Self-description
- Tunneling on link-layer
- Tunneling on busmonitor layer
- Routing
- Device Management

KNX IP The KNX IP protocol is based on IP multicast, and in its implementation equivalent to KNXnet/IP Routing. Using a multicast-based technique, KNX IP does not support busmonitor layer, and one-to-one connection establishment as required by KNXnet/IP (client to server) is not necessary.

Process Communication

With process communication, common tasks are:

- DPT encoding and decoding of Java and KNX data types
- Reading and writing process communication data
- Group monitoring

Supported Datapoint Types For process communication, Calimero implements the majority of DPTs commonly used by KNX sensors and actuators. The supported types consist of the DPTs with main numbers 1 to 20, as well as main numbers 28, 29, and 232. See table 2.1 for the list of supported DPTs.

Busmonitor

With tunneling connections on busmonitor layer, monitoring a KNX network is supported using KNXnet/IP, KNX USB, and FT1.2.[1]

Raw Frame Decoding Support for decoding a raw frame on the communication medium allows busmonitoring software to extract information from a received frame. Calimero provides decoding for following KNX communication media:

- TP1
- KNX IP
- PL110
- PL132
- RF

1 For KNX RF USB, Calimero provides an implementation; the implementation itself is not tested due to the lack of RF USB laboratory equipment with busmonitor support.

DPT	NAME	EXAMPLES
1.x	Boolean	Switch, Alarm
2.x	Boolean controlled	Switch Controlled, Enable Controlled
3.x	3 Bit controlled	Dimming, Blinds
5.x	8 Bit unsigned value	Scaling, Tariff information
6.x	8 Bit signed value	Percent (8 Bit), Status with mode
7.x	2 octet unsigned value	Unsigned count, Time period
9.x	2 octet float value	Temperature, Humidity
10.x	Time	–
11.x	Date	–
12.x	4 octet unsigned value	–
13.x	4 octet signed value	Counter pulses, Active Energy
14.x	4 octet float value	Acceleration, Electric charge
16.x	String	ASCII string, ISO-8859-1 string (Latin 1)
17.x	Scene number	–
18.x	Scene control	–
19.x	Date with time	–
20.x	8 Bit enumeration	Occupancy Mode, Blinds Control Mode
28.x	UTF-8 string	–
29.x	64 Bit signed value	Active Energy, Apparent energy
232.x	RGB color value	–

Table 2.1: Supported Datapoint Types in Calimero.

Support for Management

- KNX Application Layer Services
- KNX Management Procedures
- CEMI Local Device Management

Supported External Message Interfaces

- CEMI Standard and extended L-Data
- CEMI Busmonitor
- CEMI Device Management
- EMI1/EMI2 Standard L-Data
- EMI1/EMI2 Busmonitor

2.6.3 Network Buffer

A state/command-based datapoint buffer to answer L-Data requests, and buffer incoming L-Data indications.

The Calimero network buffer locally reflects the most up-to-date state (or command sequence) of KNX datapoints. It does this by temporarily storing KNX network messages. Reasons to use a network buffer are to enhance response times on answering frequently occurring application queries, enhance KNX network performance, and less traffic congestion on the communication medium. KNX communication media differ in processing delays and maximum message rates. This can have adverse effects, caused by queue overflows or dropped messages, for client applications that do not take those limitations into account.

Overally, a network buffer can lower KNX network load, especially in case of non-optimal access patterns to a KNX network, e.g., polling.

CHAPTER 3

Design of a Learning Platform

One important aspect of a learning platform is the well-considered integration into the overall course structure, with guidance available by supervisors where necessary. For example, as with experiments in a laboratory setup, in many cases students should not retreat to only trial-and-error strategy for accomplishing a specific task.

In this thesis, the design applies a way to allow easy sharing of the learning platforms configuration, automation setup, and collected data. This encourages students to review and discuss each others approaches. In a distributed working setup, solutions can be shared and enhanced, with each student working on her local learning platform. Improvements can be integrated, and shortcomings replaced by better versions. Alongside this, one goal of the overall design is to not introduce requirements for any additional editing software. The data structure of shareable information of the platform should be text-based, human-readable and also easily parseable by users. This is one factor to remove obstacles that could create a higher than necessary acceptance threshold for platform users.

The chapter continues with a discussion of the common and individual needs of our two target groups, pupils and graduate students. Subsequently, the design arguments are discussed for a user-oriented platform design and the graphical user interface. Another design element relates to the two other interfaces of the learning platform, and mainly target graduate students: the learning platform configuration interface, and the platform's API for extensions. For the latter point, thoughts are given to an appropriate selection of the programming language, Java for that purpose. The chapter finishes with describing common user interface (UI) toolkits and frameworks for Java.

3.1 Common & Individual Needs of Target Groups

The learning platform addresses two main groups of students, having different age groups, previous knowledge, and requirements on the platform. Hence, it is of advantage for further design decisions to start with a differentiation of needs by target group.

3.1.1 Junior Mode

With children's university activities, children participate in science and research in the various fields. Laboratories host physical setups of control systems and automation equipment, to see networked components in operation, and gather practical experience.

This mode addresses question of *how*, e.g., *How can I automate a room?*, or *How does smart metering work?* It allows participants to learn about automation in an age-appropriate way outside the laboratory.

The approach of this thesis for realizing a *junior mode* is to model any activity of the learning platform to maintain a high level of attention, so to provide entertainment value. To develop this idea for the area of domotics, the prototype will offer *quests*, consisting of goals that can be reached by understanding the effects of user decisions on a given optimization problem. Instead of a strict systematic approach as applied in optimization theory, the user will learn about the effects in a more playful manner. Selected indicators will provide feedback on decisions with respect to achieving the goal.

Quests

The following two examples are provided to show how optimization problems can serve junior mode by being realized as quests.

Energy-efficient Living Optimize the power consumption of a building or apartment, under the requirement that the living environment has to be equipped to a certain comfort level. The gaming character comes from indicators that provide the user with both an abstracted energy efficiency level of the building, as well as a saturation level reflecting the completeness of installed appliances.

A Storm is Coming! Prepare a building or apartment for an upcoming storm. Starting with a storm warning, containing a weather update, the player gains useful information about the quest. She then tries putting together the necessary building blocks of automation and control for the property, allowing the resulting logic to accordingly react to the predicted weather conditions, protecting the building. The gaming character comes from establishing a time limit, using a count down until the storm hits the location.

3.1.2 Graduate Mode

A graduate level student who uses the learning platform will mainly focus on creating automation setups, in order to understand and execute a laboratory assignment. Therefore, the platform support aims at the predominant tasks in automation and control. In the following, the main topics of activity are described for graduate mode.

Communication Protocols

By providing implementations of different communication protocols, a student can compare differences of the applied protocols, and evaluate their advantages and limitations for a given use-case.

Logic Building Blocks

Starting with single components provided by the platform in form of sensors, actuators, and controllers, a student is able to build more complex logic with domain-specific behaviour. Those are supported by grouping and connecting device functionality over the network to form distributed services. Examples are lighting and shading logic for a room, or smart metering an apartment.

Causality

Establish and execute the required steps necessary to generate the sequence of messages for executing a single action within a control loop of an automation system. One example is the realization of a simple dynamic system: apply controller logic to execute an actuator, where the actuator's output behavior follows sensor feedback, to satisfy a given objective, i.e., specified by a set point value or the *reference* signal.

Representation

Provide information of transmitted frames in a specific network segment. This is accomplished by recording the messages sent on a dedicated link, and dissecting the recorded messages. For the exchanged messages being passed through the network stacks, the possible range covers data link layer to application layer representations. With respect to representing the message data, the possibilities range from formatting bit-level data to interpreting the field semantic of messages, showing human-readable output.

Evaluation

Support the evaluation of networks characteristics by implementing user access to specific performance metrics of a subnetwork. Useful metrics for a communication link include the total bits transmitted and transmitted bits per second, as well as message count. For a gateway, a useful metric is the number of filtered link layer messages. In this case, the metric could serve as performance indicator of an applied group address filter table for a specific subnetwork. For evaluating how energy-efficient a setup is, metering the power consumption of devices attached to a subnetwork can help the student along further decisions.

3.1.3 Common Needs

This section discusses general platform features for teaching, experimentation, self-studying. Those features apply to both junior and graduate mode, and can be seen as common needs of the target groups.

The approached learning platform should include features to address the following three situations during the learning process.

Taught Concepts

To maximize its didactic purpose for teaching, a platform design is desired that allows to create automation setups based on the taught concepts of distributed control and automation. Specifically, this includes the ability to recreate physical setups in the platform for evaluation. This also holds true for junior mode: the ability to resemble and relate to physical setups is of advantage, e.g., setups as known from the own apartment, or seen during "open house" laboratory activities.

For the main purpose of the platform – communication protocols and network communication – this includes the ability to apply and visualize group-based communication patterns, and communication with individual or broadcast destinations. In KNX, group-based communication is used for process communication, with data exchange of KNX datapoints consisting of state- or command-based datapoint values. Unicast and broadcast patterns mainly apply to management tasks for devices, or network management procedures.

Experimentation

By conducting experiments with the learning platform, students should first of all gain insight into cause-and-effect when altering communication-related parameters, i.e., of the selected communication medium or communication protocol.

In junior mode, this mostly refers to basic experiments like adding and removing specific devices, or creating simple building logic by grouping communication objects. In KNX, the prevalent way to do this, is to let devices communicate via *datapoints*.

For experimentation primarily focused on graduate mode, the goal is supporting students performing their assigned tasks, similar as for a physical setup in the laboratory. Tasks are solved by not only changing configuration and communication factors, but also by creating the experimental setups – individual control and automation setups – from scratch.

Another area for experimentation is to enhance or introduce new device models and specific implementations thereof, for subsequent evaluation in an existing automation setup. In the end, those implementations extend the learning platform, but involve certain programming efforts on the student's side. Hence, the platform should feature an API that allows a student to interface with, and extend, the device model. This interface is best kept above application layer in the OSI model ("user layer").

Self-studying

Self-studying is the activity of a student to study outside the laboratory, without direct supervision. This applies to further reading material provided throughout a course by the lecturer, but also covers activities like educational games.

To include the learning platform in self-study activities, the first approach would be to browse the GUI-supported configuration possibilities, and use message dissectors. Message dissectors, introduced in section 3.1.2, split a message into its various fields and decode the content for further examination. A subsequent step could involve studying the configuration files and templates bundled with the learning platform, for configuring the various device types and the communication network.

For educational games, *quests* are described for junior mode in section 3.1.1 on page 22. Those quests can be adapted to graduate mode as well, by providing more complex challenges. In the end, the overall approach of supporting a *serious game concept* in the learning platform applies to both junior and graduate mode.

3.2 Platform Design

Examining design aspects, this section first specifies the overall architecture of the system. Subsequently, this section discusses the primary design elements that the platform design distinguishes from, and shares with, other types of laboratories.

3.2.1 System Architecture

The system architecture of the learning platform is based on the model–view–controller (MVC) pattern. Model–view–controller is a pattern common in software architectures, and describes a separation in terms of responsibilities. Each of the three components – *Model*, *View*, *Controller* – captures one responsibility:

- Model – responsible for behavior in the problem domain
- View – responsible for behavior in terms of output representation

- Controller – responsible for interaction with the user in terms of commands or actions

In comparison to the similar model–view–adapter (MVA) pattern, the MVC pattern allows direct communication between the view and the model. For the learning platform, such direct communication is useful to initialize the view to a pre-known state, and presentation of model data. Examples are KNX datapoints, state or command-based values, and spatial layout information. That information can be either directly or indirectly specified by the user, therefore, does not require interception of controller logic.

In contrast, the MVA pattern would require to reschedule the execution consisting of the sequence of management and process communication events to arrive at the expected state, i.e., the last output representation known to the user.

3.2.2 Primary Design Elements

The concept of a learning platform as proposed in this work distinguishes itself in several elements from other approaches in software, but also other forms of laboratories. This also manifests in the overall platform design.

First, the platform is designed with the goal of a low acceptance threshold by users. This specifically aims at interested younger students and attendants of introductory courses at the university for topics on distributed automation and communications. At the same time, this also puts certain limitations on user interfaces. The user interface will not provide configurations that only be managed with enhanced property dialogs. Optimally, no initial configuration is required for basic operation. Similarly, the platform will not contain the sometimes surfacing "expert mode"-switches for *very* advanced users. Usually, such a button is just a carrot most cannot withstand to crunch. Advanced configuration is maintained in structured configuration files.

The platform recognizes the increasing replacement of traditional workstations with smart devices, e.g., tablets. See table 3.1 for an analysis and forecast of the total shipments worldwide by device types. Therefore, the overall design accounts for a cross-platform development. This includes the different computer platforms commonly used by the target groups, and allows deploying the learning platform software to those platforms.

The didactic focus of the learning platform is to introduce the concepts of communication protocols and show the functioning of a device network stack. To provide a realistic learning environment, those parts are not simulated, but a network stack is used as also implemented in physical KNX devices. This excludes introducing inaccuracies of a dedicated simulation model, and avoids unwanted divergency through abstraction.

Device Type[1]	*2014*	*2015*	*2016*	*2017*
Traditional PCs (Desk-Based and Notebook)	277	247	234	226
Ultramobiles (Premium)	37	44	57	78
PC Market	314	291	291	303
Ultramobiles (Tablets and Clamshells)	226	199	208	218
Computing Devices Market	540	490	499	521
Mobile Phones	1,879	1,905	1,960	2,000
Total Devices Market	2,419	2,395	2,459	2,521

Table 3.1: Totals of worldwide device shipments by device type, in millions of units [38, Table 1].

The visualization of the GUI takes advantage of modern design features supported by user interface toolkits and frameworks. Sensors and actuators resemble physical devices, and use animations for graphical feedback. The modeled system environment is also visualized using animations, e.g., using local weather data.

A target group of the software design are younger students that participate in children's university activities. Hence, a design is preferred that balances the use of design elements with a pure technical or configuration purpose in the GUI. Advanced concepts, e.g., device management, are supported via KNX communication services or programmatic access, and rather aim at graduate students.

The GUI design is kept simple, and the overall interaction with the interface should be intuitive to a user. In part, this is owed to the fact that user behavior does not meet the traditional expectation of studying a manual for assistance before using a system. A simple design also supports two of the elements already discussed: to address users at a younger age, and users with little to no detailed knowledge about distributed automation systems. And second, the deployment to user platforms, including smart devices. Implementing complex and unusual ways of interaction, especially on multitouch-enabled devices, is not constructive for user motivation.

Aside from the distinctive aspects, some elements are shared with other approaches. With simulated laboratories, the design shares the faster and safer prototyping and development functionality. No laboratory hardware is required, the device types are implemented in software. As with remote laboratories, no physical attendance is re-

1 Notes in [38] for the device categories in table 3.1: "The Ultramobile (Premium) category includes devices such as Microsoft's Windows 8 Intel x86 products and Apple's MacBook Air. The Ultramobile (Tablets and Clamshells) category includes devices such as, iPad, iPad mini, Samsung Galaxy Tab S 10.5, Nexus 7 and Acer Iconia Tab 8. Ultramobiles – All Ultramobile Basic and Utility Devices."

quired. Still, the student has access to an accurate representation of communication protocols and network stacks for experimentation.

3.3 User Interface

3.3.1 Dialogs

The approach in this work is to avoid dialog-based interaction, in particular modal dialogs, or minimize their usage to the least extent possible. Depending on the window manager, multiple modal dialogs can stack or queue up. Each of the dialogs has to be closed before the actual work can continue. A less disruptive dialog is a foreground dialog. It stays top-most in the window z-order of the owning application, but does not disable interaction with other windows of the application.

3.3.2 Notifications

To avoid the negative aspects of modal or foreground dialogs, various forms of popup windows are a good alternative. The main advantage is that such a window component does not disrupt the user workflow.

Popup Component

For prototyping the learning platform, the GUI design is heavily based on various forms of a popup component. The following paragraphs shortly describe the main configurations of popups implemented in the learning platform software.

Content A popup component can have different content. In its simplest form, a text string is shown, optionally also in form of a hyperlink. Enhanced versions can have buttons to interact with the popup, usually a close or hide button. Commonly used to aid user comprehension is the addition of pictograms ("icons") as visual cues, consisting of simplified or stylized pictures, or usage metaphors.

The design of the learning platform will also use more complex layouts for the content to show device configuration, communication settings, and messages. For that, table controls are used. This allows, e.g., to show a message sent by a device, together with its dissection by communication layer, and a human-readable decoding of various message fields. Another use-case of a table is to list the assigned addresses and datapoints of a device object, e.g., a sensor. The goal is to minimize additional property dialogs and tabbed pane layouts.

Timeout An instance of a popup component can be initialized with a timeout, to automatically close itself at when the time expires. If the instance is closed earlier by other means, e.g., a close button, any scheduled event for the timeout condition is canceled.

Balloon Tip A configuration in form of a balloon tip shows input hints, status and usage information, or recommended user actions. It is usually realized as a popup window control that is placed over the working area using a certain horizontal and vertical offset to its specific target.

In general, a balloon tip can be associated with any window control, but also with rectangular areas within a specific control. This allows, e.g., to show tips for a specific table cell.

Tooltip A slight variation of the popup component is the tooltip. A tooltip is associated with one or more window controls, and initialized with an (optional) time delay. On hovering over any of the respective window controls, the tooltip shows up after its delay expires. If no initial time delay is set, the tooltip shows up immediately. A toolip is closed either by using a timeout, the hover operation finishes (e.g., the mouse pointer is no longer over any of the associated controls), or the mouse pointer moves into the bounding box of the tooltip control. While useful when interacting with the GUI using pointing devices like a computer mouse or touchpad, on multitouch-enabled devices tooltips are insignificant as design decision.

For text controls, a common alternative to tool-tips is to show the text message as input hint directly in the text field. As soon as actual characters are entered, the input hint is removed. In HyperText Markup Language (HTML), such hint is also supported and known via the *placeholder attribute*. The only minor disadvantage of that approach is that if a text field is not empty, e.g., set to a default input, the input hint is not visible to the user anymore.

3.4 Programming Language Selection

The following three points are the main contributors in the language decision:

- The implementation language of essential libraries to provide communication protocols and networking functionality
- Availability of UI toolkits and frameworks for the GUI for a specific language
- Possibility of deployment to common desktop and mobile platforms

A single language might not necessarily fully cover all points listed above. If necessary, deciding on a different language for different MVC components is also an option.

In the MVC pattern, the model component is responsible for behavior in the problem domain. For this thesis, the model's problem domain is the KNX network stack, communication network, and KNX devices. Calimero, the selected library for KNX communication is implemented in Java. Therefore, any necessary code adaptations will also be in Java. As an alternative, other languages that maintain language interoperability to Java can be used, e.g., Scala [39].

An important part of point three, deployment, is choosing a language that supports software development agnostic to the computer architecture, without recompilation on a target platform. This addresses both target groups of this thesis. Pupils usually do not have software engineering background and access to working setups for compilation and installation. Students working in several different environments at the laboratory, university, and home typically face different platforms and operating systems. Minimizing deployment overhead avoids problems and frustration during initial installation and setup.

The view component is tightly coupled to the available UI toolkits and frameworks. A main decision to make is whether to use a web-based user interface, or a traditional desktop/mobile interface. Main candidates for deployment as web application are frameworks and libraries based on HyperText Markup Language Revision 5 (HTML5) [40] for content description, using JavaScript [41] logic and Cascading Style Sheets (CSS) [42] for presentation.

With the user interface being prototyped as desktop (and mobile) application, the approach with the most synergies would be to stay within the same language. In this case, Java, or any interoperable language like Scala.

A noteworthy toolkit for web-based application front-ends is Google Web Toolkit (GWT) [43]. GWT allows implementation and debugging in the Java language. The GWT cross-compiler compiles the Java source code to JavaScript for deployment as web application. In addition, the source code can also contain JavaScript. For mobile deployment, GWT together with HTML5 can embed different views appropriate for the target platform. This interesting approach makes it a viable candidate for using the Java-based model to develop a web application. One serious drawback, which finally led to its exclusion are the difficulties arising in cross-compiling the Calimero library. Being a network library, most of the Calimero protocols interact with network devices, and specify timing constraints. The intended workflow of automatic cross-compilation terminates with errors, making several manual adaptation necessary by the developers. The unit-tests are written in Java using jUnit [44]; hence, all required modifications to the source code for successful cross-compilation can no longer be tested.

Using a UI toolkit written in a language like C++ would either require individual deployment for each supported target platform, or recompilation of the UI part. The same applies when implementing the UI by using native UI components of the operating system. In the end, this would effectively diminish the advantage of using an "architecture-neutral" language like Java in the first place.

Considering all discussed points, the best matching programming language, and therefore, the language selected for implementing the learning platform software and programming the user interface, is Java. Section 3.5 continues with a short description of common UI toolkits and frameworks for Java.

3.5 User Interface Toolkits and Frameworks

This section covers toolkits and frameworks for Java commonly used in creating GUI-based applications. The selection criteria is on software that is *a)* provided as either free or open source software, or *b)* distributed as part of development environments or language libraries.

3.5.1 AWT and Swing-based Implementations

- Abstract Window Toolkit (AWT) [45] was the first GUI toolkit included in Java, providing a thin layer on top of the underlying platform's native controls. In contrast to other toolkits listed here, certain profiles of the Java ME runtime require AWT for their user interaction. By today's standards, AWT is not considered an efficient toolkit for Rich Client Platforms (RCPs).

- Swing [46] has its foundation in AWT, but offers many additional complex components. Swing is designed around a pluggable look and feel (PLAF) mechanism, which gives the developer a choice of deciding the "look and feel" of the GUI, also at runtime. With the advancement of user devices new forms of interaction, Swing lacked support of new features, e.g., multi-touch interaction, UI customization, or animation.

- SwingX [47] provides extensions to the Swing GUI toolkit, and is by nature heavily based on Swing. The main goal was to enrich the set of available components, including highlighting, auto-completion, support for tree tables, besides others.

3.5.2 Other Implementations

- The Standard Widget Toolkit (SWT) [48] is a toolkit that provides access to the operation system's native GUI components. The development of SWT was started based on an initiative of IBM, with the first software using a SWT-based GUI being the popular Eclipse framework. SWT is licensed under the Eclipse Public License (EPL).

- JavaFX [49] is promoted by Oracle as successor to Swing. The user interface can either be programmed in Java or specified in FXML, an Extensible Markup Language (XML)-based language. Using FXML provides a clear separation of the user interface structure from other aspects of the software (which aids the MVC concept). The look and feel of the JavaFX graphical presentation can be specified using CSS. Starting with Java 8, JavaFX controls use the Modena CSS for their UI. JavaFX has the same license as Java SE.

- Apache Pivot [50] has its focus on Rich Internet Application (RIA) and desktop applications. Apache Pivot relies on Java2D for rendering.

- Qt & Qt Jambi – Qt [51] is a development framework for both the creation of applications and user interfaces. It is written in C++ using a preprocessor extension called Meta-Object Compiler (MOC), that adds features like a signal and slots communication mechanism.

  The framework aims at cross-platform development for desktop, embedded, and mobile platforms, providing its own Integrated Development Environment (IDE) and build system. The creation of cross-platform GUIs is supported via the Qt Widgets module, or the QtQuick module. QtQuick allows the creation of animated, touch-enabled interfaces. It is based on Qt Meta-Object Language (QML), a declarative language to describe the user interface with JavaScript as scripting language. Qt offers community licenses using several versions of the GPL and LGPL.

  Qt Jambi is a Java binding of Qt for use by Java desktop applications [52]. It benefits from Qt's cross-platform abstraction and native look and feel of widgets on supported platforms, and therefore, can offer many GUI components to Java developers.

3.5.3 Notes on Implementations

One aspect with GUI toolkits is that many are not thread-safe. Logic that interacts with UI components often has to execute on a single distinguished thread for that purpose. Of the described toolkits and frameworks, the implementations among the popular ones like Swing, SWT, or JavaFX, all implement a dedicated event dispatcher thread used for GUI events, and mandatory for interaction and event handling.

Emulation of UI Components

Toolkits that provide their own user interface components, ranging from simple components like text fields or check boxes, to complex ones like tree-views, tables, or file dialogs, do almost never match (and also cannot exactly match) the components native to the Operating System (OS). This can be by design, with the goal to instead provide the user a uniform experience across different platforms. But in many cases, even subtle differences when switching between native and "emulated" components cause confusion. This is especially true with users less computer-oriented, e.g., elderly people or children. An improvement is the use of OS theme libraries and rendering engines where possible to draw the toolkit components. This ensures a closer match of the look and feel. As the overall design of the learning platform is not heavily based on controls for RCP, the decision of native versus emulated components will not cause conflicts with the overall design.

3.6 Analysis Methods

Two methods are applied for analysing the resulting platform design.

The A-Lab at the Institute of Automation contains a physical setup of an with equipment commonly found in a automation setup. The equipment consists of both sensors and actuators supporting the KNX standard. The first method relies on an accurate representation of the physical setup in the prototyped learning platform. Accuracy is defined and determined in the following ways:

- Support of sensors and actuators: the set of common equipment is available in the prototyped solution.
- Support of communication with the prototyped devices in the same way as possible in the physical setup. Main focus is on process communication and simple device and network management tasks.
- Representation of the equipment in the prototyped solution: does the output representation (view component) provide sufficient information to operate the prototype in junior and graduate mode. Because graduate mode is a super set in functionality of junior mode, determining representation can mainly focus on graduate mode.

The second method is based on an iterative qualitative approach of user feedback. The aim of the learning platform for graduates is to support teaching of KNX communication principles, and allow laboratory exercises without physical equipment. The required feedback is gathered by letting students replay tasks on the prototype that were executed successfully on the physical setup. The user experience is evaluated in a comparative way between physical setup and software prototype.

The aim for juniors is to introduce automation concepts in an age-appropriate way. Therefore, evaluating the user experience is focused on a comparison with educational games the participant like to play, gradients of motivation and subjective interest in the topic, as well as easy interaction with the UI.

Both types of feedback are used to distill possible feature improvements to the learning platform. An important point to note here is that laboratory setups and are subject to modifications and updates over time. Hence, applying the results of an analysis has as goal providing a feature complete prototype. Nevertheless, it can also indicate stability and performance improvements.

CHAPTER 4

Model and Operation

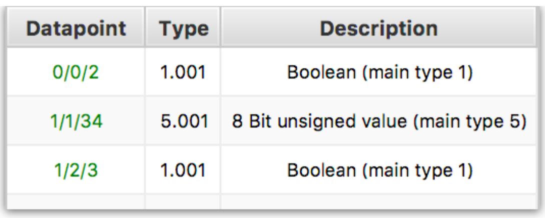
4.1 Models

The problem domain, according to the MVC architecture contains several models for abstracting specific behavior of the learning platform.

4.1.1 Communication Model

For this model, behavior is mostly specified by the KNX specification, and encapsulated by the selected Calimero network libraries.

Part of the communication model is the use of datapoints for process communication. Datapoints can either be configured via configuration files, programmatically using KNX application layer services, or in the GUI for a device object (see figure 4.1).



Datapoint	Type	Description
0/0/2	1.001	Boolean (main type 1)
1/1/34	5.001	8 Bit unsigned value (main type 5)
1/2/3	1.001	Boolean (main type 1)

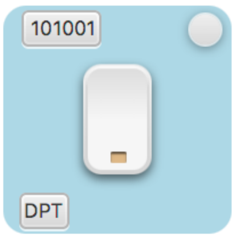
Figure 4.1: Device popup, listing the assigned datapoints for process communication.

4.1.2 Device Model

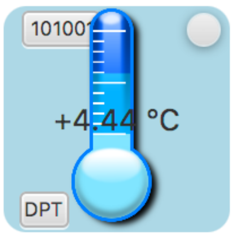
The device model covers sensors, actuators, and controllers. A selection of sensors is shown in figure 4.2, with figure 4.3 showing a selection of meter devices. Figure 4.4 shows a selection of actuators.

The device model provides two states for use with the GUI: *inactive* and *active*. At platform startup, one object of each available device type is placed as a representative of its device type on a dedicated bar in the GUI. Initially, each object is in inactive state. Such inactive object serves as icon, to be dragged and dropped onto the floor plan by the user. On the start event of the drag operation, a copy of the selected device icon is created and animated to follow the mouse cursor to the intended drop location. On the drop event, the dropped object is activated. When put into active state, a device will start its operation – sensing, actuating, or controlling – according to device type. Figure 4.5 shows the popup of an oscilloscope in active state.

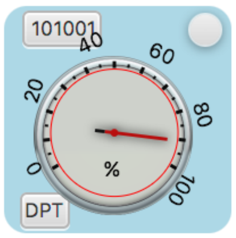
A device provides an *action* method to perform its operation, triggered by external events as well as communication events. The following examples should show its application in specific realizations of the device model. In the case of a temperature sensor, that operation would be transforming an external event – change in temperature – into a status update of the communication object representing the temperature value, as well as updating the representation of the sensor in the GUI. For a mercury thermometer sensor, the update would involve adjusting the liquid of the capillary glass tube.



(a) Two-state push-button with status

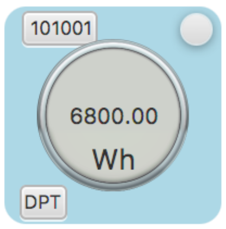


(b) Temperature sensor

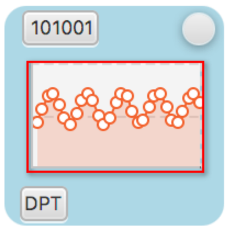


(c) Humidity sensor

Figure 4.2: A selection of sensors in the device model.

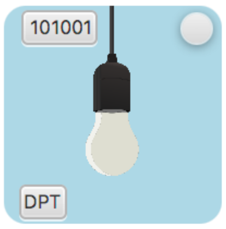


(a) Smart meter

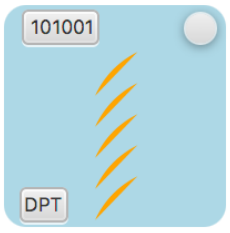


(b) Oscilloscope

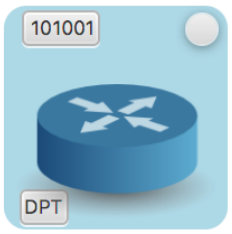
Figure 4.3: A selection of meters in the device model.



(a) Light bulb



(b) Venetian blinds



(c) Home gateway

Figure 4.4: A selection of actuators in the device model.

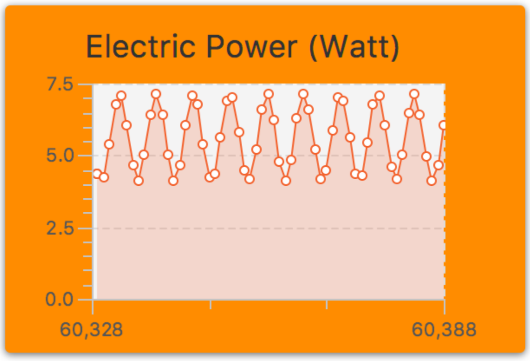


Figure 4.5: An oscilloscope popup, in this example for electric power (Watt) over time.

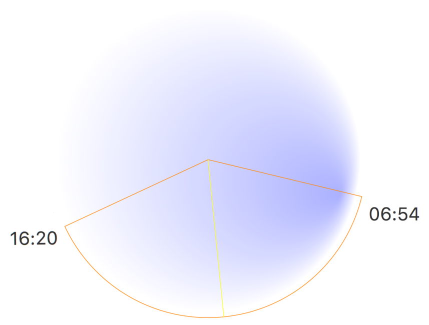
4.1.3 Weather Model

One model deployed in the platform applicable to certain sensor types is the weather model. The aim is to provide realistic sensor input data based on information about weather conditions, wind, atmosphere, and astronomy. More specifically, provide input data so that the state values from a set of sensors are coherent for configuring, executing, and reasoning about control strategies. Figure 4.6 shows the visualization of the day arc of the sun and sky color in the weather model at different times during the day. For further details on model metrics, refer to section 4.2.1.

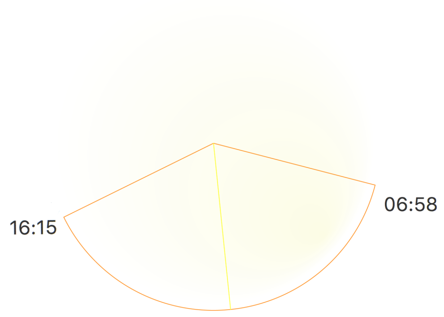
4.1.4 Energy Model

One modeling aspect of the learning platform relates to energy management and smart metering using KNX. The purpose of a smart meter device deployed in a building is to record consumption of electric energy in certain time intervals and report the collected information. One use-case is billing, another one monitoring power consumption.

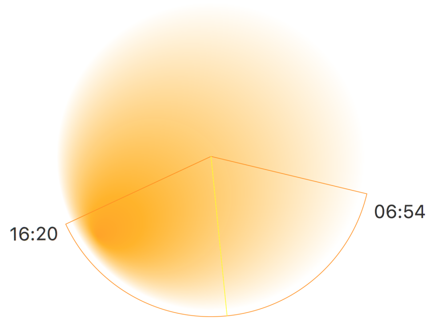
To support that modeling aspect, the modeled consumer devices in the building should be able to report back consumption.



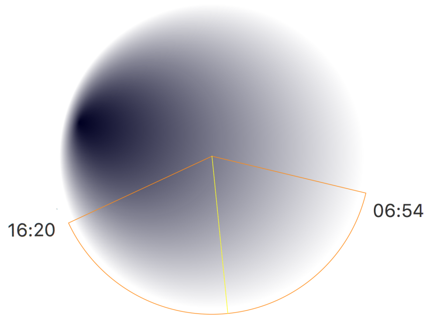
(a) Sunrise



(b) Forenoon



(c) Sunset



(d) Night

Figure 4.6: Portrayal of the day arc of the sun, with time markers for sunrise and sunset (circle segment), zenith (yellow line), and sky color gradient depending on day time.

The specifics of usage distribution, consumption profiles, and how particular appliances calculate their consumption are complex and outside the topic of this thesis. As are the various levels of abstractions in designing an appropriate energy model. For the purpose of the learning platform, a very simple approach is sufficient to support the mental model of students about energy management tasks. For clarity, that model is limited to electrical energy. The energy model makes a coarse distinction between background electrical loads which are considered constant loads, and flexible loads.

Flexible loads in the model can either be described via equations, provided in the platform's configuration of a device and evaluated during runtime. As an alternative, a flexible load is described by playing back a trace recorded from a matching appliance. The prototype supports the following ways of specifying a load:

a. Expression-based, e.g., $3 \times sin(t)$, with t being the time variable

b. Derivative loads, where the load is calculated taking the derivative of a configured parametrized equation

c. Time series stored as trace, i.e., a sequence of data of successive measurements over a time interval

Evaluation of all loads during runtime and summation over all connected devices then provides the building load curve. That curve can be used for smart metering. A smart meter device type in the learning platform receives updates on any load change calculated by the energy model. A smart meter object visualizes the load in its graphical representation, but the load value can also be queried using communication services, i.e., using process communication to read from the corresponding smart meter datapoint. The power network itself is not modeled.

Building upon such a simple energy model, enhancements could allow to study the effect of deploying complex control strategies, e.g., scheduling problems like load management schemes for flattening electricity usage.

4.2 Collection of Information

A useful set of information for the learning platform can be gathered by knowledge about the current meteorological situation. With such knowledge, the platform can align its model of sensor input, providing a better approximation using current conditions in a specific location. Therefore, the prototype implements query support to online weather services.

4.2.1 Weather Model

For the platform prototype, two weather APIs have been selected, based on the provided data response, weather update interval, and terms of usage. One requirement is free service for non-commercial use; both APIs offer that. The first choice is Yahoo Weather API [53], followed by OpenWeatherMap [54].

Both services provide weather data for any location on Earth, using Representational State Transfer (REST)-based queries addressed to the weather service endpoint.

Yahoo provides weather data by submitting a request using Yahoo Query Language (YQL), a Structured Query Language (SQL)-like query language. A location is referenced using a Where On Earth ID (WOEID), a 32-bit identifier of a spatial entity.

An example of a YQL query for Vienna, Austria, requesting all available information looks as follows:

```
select *
from weather.forecast
where woeid in (
        select woeid
        from geo.places(1)
        where text="vienna,␣at")
```

Using an asterisk ⋆ in a query specifies to return all columns of the queried table.

An interesting information for the learning platform are the sunrise and sunset times. Replacing the ⋆ of the *select* statement in the YQL query, only such specific information can be obtained, e.g., by forming the REST request using

```
select astronomy.sunset
from weather.forecast
where woeid in (
        select woeid
        from geo.places(1)
        where text="vienna,␣at")
```

OpenWeatherMap supports lookup of a city location by city name and country code, city Identifier (ID), geographic coordinates using latitude and longitude, or ZIP code. The country code expected according to the ISO 3166 country codes. For example, a lookup query for Vienna, Austria is written as (without quotes) "`api.openweathermap.org/data/2.5/weather?q=vienna,at`" . OpenWeatherMap has the benefit of multilingual support, including several European languages.

Most weather APIs have the option of several response formats. Common ones are JavaScript Object Notation (JSON), XML, and HTML. In our case, the selected response format is JSON. For reasons of bandwidth and fairness, most providers state a rate limit, specifying an upper request volume within a given duration. For example, with Yahoo the rate limit is 2,000 signed calls per day [53, in 2015]. Those rate limits might affect widely distributed applications, it is of no further concern for the learning platform prototype. In the prototype, data requests are by default issued only during startup, and then every 30 minutes.[1] And in case, a fallback mechanism can switch to the alternative service provider.

Interestingly, the Yahoo API works also without an API key. By default, a user requires a consumer or API key for identification. (OpenWeatherMap does reject queries without a valid API key.)

In general, an API response – if not already specifically constrained in the corresponding request – contains a multitude of information about the current weather condition at the requested location. For the purpose of the learning platform, of immediate interest are the ones listed in table 4.1. The table shows a categorization of different metrics

1 Using Yahoo weather data, 20 students could each start the platform 25 times per day, each instance running for < 2 hours wall clock time.

Category	*Attribute*	*Application in sensing and control*
Condition		
	Temperature	Openings, shading, HVAC, energy management
	Min. Temperature	HVAC (long-term strategy)
	Max. Temperature	HVAC (long-term strategy)
	Description[1]	Human-readable notification
Wind		
	Direction	Wind-facing facades of a building's envelope
	Speed	Threshold and severity of control strategies
Atmosphere		
	Humidity	Openings and HVAC
	Visibility	Shading
Astronomy		
	Sunrise	Day/night modes, lighting, shading, energy mgmt
	Sunset	Day/night modes, lighting, shading, energy mgmt

Table 4.1: Meteorological metrics useful in a model providing sensor input.

and their possible application by a model providing data for sensor input and control decisions, e.g., for HVAC.

Similar information is provided when using the weather API requesting a weather forecast.

4.3 Time

A complete setup created in the learning platform contains the structural information, consisting of the deployed sensors, actuators, and controller logic, as well as deployed data models, consisting of configurations for weather, energy, and communication.

To evaluate such automation setup demands for a certain minimum duration of time. The models for sensor input, i.e., weather, energy, have only effect on building control and automation when executed over a longer period of time. This is due to their nature of aligning to a realistic representation: weather conditions usually do not change abruptly, and the variations in energy consumption are limited by the installed consumers in a household.

By default, the learning platform executes in accordance to computer time, i.e., *wall clock* time. This also implicates that any time-dependent sequences in communication

1 Examples are *cloudy*, *showers*, *foggy*, *heavy drizzle*, etc.

protocols are unaltered and execute in a similar time fashion as compared to any physical setup.

To allow evaluation of the automation setup within a shortened amount of time, the learning platform provides the user with functionality to speedup execution. This functionality is implemented by introducing simulation time in the platform models. Simulation time is provided by scaling wall clock time with a constant factor.

Hence, the new execution time is proportional to the old execution time by the specified speedup. This relation can be written as speedup $= T_{\text{wallclock}}/T_{\text{sim}}$, so the resulting scale factor for time is $1/\text{speedup}$.

The maximum speedup that can be achieved is limited in practice by the lower time bound required to update the system state and views of the learning platform. Although that bound is a soft bound, many violations during execution of the platform models' contrast with the original evaluation purpose. The scaled time constants of the communication protocols will become that small to cause message timeouts and omission faults. Although many of the communication protocols specify a message resilience degree > 0, at such point the observed communication behaviour starts to deviate from expected behaviour. The result cannot be considered accurate anymore.

4.3.1 Non-proportional Speedup

For providing evaluations of night/day controller logic, or similar durations in the range of several hours, a proportional scaling of all time constants does not prove successful. The reason is the necessary small scale factor to accommodate for an appropriate short time duration of a single evaluation run.

Scaling timeout constants in communication protocols that are predominantly in the time range of several hundred milliseconds to a few seconds leads to an increase in timeout violations and message omissions. This is mainly caused by mechanisms and activities of the underlying operation system and system libraries, accumulating a (non-deterministic) time delay to the processing time of protocol executions. Examples are thread scheduling, time slicing algorithms, queuing delays, and optimizations in interacting with the GUI.

Under the assumption that time-based variables of small range in the various models are mostly confined to a specific model and independent across models, multiple scale factors can be used and not negatively affecting evaluation. Those additional factors scale quantities in the learning platform non-proportional to simulation time, i.e., the time obtained by scaling wall clock time. As already indicated, of specific interest is the application of such non-proportional scale factors in the communication model. Animated components in the GUI are another use-case of non-proportional scale factors. The main purpose of such animations is to provide user feedback with respect to an event or action, e.g., tilting venetian blinds. Keeping a certain minimum duration for an animation, e.g., five seconds for our example of tilting, do not introduce adverse effects on evaluation results.

4.4 Integration of Communication Protocols

For the integration of knx communication protocols, the following two communication domains exist in the learning platform: 1) protocols executing within the building's communication network (building domain), and 2) access protocols to the building's network from endpoints outside the building communication domain. The second domain interfaces with the building domain via a gateway.

4.4.1 Nodes and Building Network

The network stack of a KNX device will for most parts try to pass message information in a medium independent manner. At the link layer, a decision of the EMI takes place. The selected format depends on the supported EMI formats of the KNX interface, e.g., the BCU. In case of CEMI, a message can be enhanced by medium-specific information using dedicated *additional information* fields, e.g., to accommodate the RF medium.

But the differences in frame formats of the various communication media is also influenced by communication protocols. Using KNX IP requires CEMI only, while a TP1 link using the TP1 communication protocol – to no big surprise – requires a TP1-specific format.

To allow studying those differences in formats, the building communication network of the learning platform should be configurable. The actual transmission on physical layer is not a crucial factor. The focus is on visualizing the network with its configuration and messaging formats. For example, using power-line communication does implicate power-line specific medium settings, but not an accurate model of the physical transmission medium. Hence, for the initial prototype a realization on link layer with the possibility to examine messages on link layer is sufficient.

Using this abstraction, virtual links can be used to realize a building communication network.

Relying on the supported communication protocols and medium configurations of Calimero, the following configurations are possible for the building's network:

a. TP1

b. KNX IP

c. PL110

d. RF

KNX IP provides a special case for realization. Most target platforms provide an Ethernet interface and a virtual loopback interface. Through such an interface the modeled KNX devices can communicate on the same machine. Hence, any KNX IP messages that are multicasted on the configured KNX IP building communication network, are directly passed back up the network software stack. A realization as described does not require any link virtualization.

By extension, selecting any of the possible network configurations will make the network stacks of the connected KNX devices act as part of that selected communication medium.

4.4.2 Gateway

A user of the learning platform is provided with three different ways of interaction: *1)* via the GUI, *2)* modifying the configuration loaded during application startup, and *3)* using external communication software to access the platform.

For external communication, the platform provides a dedicated node to interface with the user. That node is referred to as *gateway*. In the prototype implementation, the gateway is realized using a KNXnet/IP server. With such gateway for communication, the user acts as the client, with the gateway providing KNXnet/IP server functionality.

Using KNXnet/IP aligns well to existing physical setups, where KNXnet/IP protocols are commonly used to provide remote communicate access to a KNX network. Therefore, this approach enhances the educational purpose of the learning platform. An example of a KNXnet/IP server configuration is shown in listing 4.1. The configuration of that listing specifies that the server-side shall be reachable over the default KNX User Datagram Protocol (UDP) port 3671 on the host's `eth0` interface, or via KNXnet/IP Routing. The KNX subnet uses the KNX IP communication medium with IP multicast `224.0.23.14`. The server will filter group-addressed frames to that subnet according to the group-address filter table, forwarding only to the destinations `2/3/1`, `2/3/2`, and `2/3/3`. Client KNXnet/IP connections are assigned any unused address from the set of additional addresses `7.1.2-4`; a client connection is rejected if all additional addresses are currently assigned.

4.4.3 Interaction with Physical Devices

Although not the learning platform's primary design goal, a specific addition allows interaction with physical devices. Initially, this interesting feature mainly evolved during development, to increase the set of different device types available for communication tests.

For KNX IP networks, a physical KNX IP device can easily be integrated in the communication setup of the learning platform by joining the same multicast group. As a

Listing 4.1: Example configuration for the Calimero KNXnet/IP server.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Calimero KNXnet/IP server settings -->
<knxServer name="knx-gateway" friendlyName="KNX Home-Gateway">
  <propertyDefinitions ref="resources/properties.xml" />

  <!-- KNXnet/IP search & discovery -->
  <discovery listenNetIf="all" outgoingNetIf="all" activate="true" />

  <!-- Provides the KNXnet/IP-side configuration for access to one KNX subnet -->
  <serviceContainer activate="true" routing="true" allowNetworkMonitoring="true"
          udpPort="3671" listenNetIf="eth0" reuseCtrlEP="true">
    <knxAddress type="individual">7.1.1</knxAddress>

    <!-- Specify the KNX subnet type, connecting the KNX network -->
    <!-- KNX communication medium: one of { "tp1" (default), "pl110", "knxip", "rf" } -->
    <knxSubnet type="knxip" listenNetIf="eth0">224.0.23.14</knxSubnet>

    <!-- KNX group address filter applied for this service container (optional) -->
    <groupAddressFilter>
      <knxAddress type="group">2/3/1</knxAddress>
      <knxAddress type="group">2/3/2</knxAddress>
      <knxAddress type="group">2/3/3</knxAddress>
    </groupAddressFilter>
    <!-- Additional KNX individual addresses assigned to KNXnet/IP connections (optional) -->
    <additionalAddresses>
      <knxAddress type="individual">7.1.2</knxAddress>
      <knxAddress type="individual">7.1.2</knxAddress>
      <knxAddress type="individual">7.1.3</knxAddress>
      <knxAddress type="individual">7.1.4</knxAddress>
    </additionalAddresses>
  </serviceContainer>
  <!-- Next service container (optional) -->
</knxServer>
```

result of doing this, a physical device is treated equally to any platform device. For the other communication media, i.e., TP1, PL110, and RF, the platform uses a proxy device. Such proxy is necessary because the listed communication media only exist virtualized in the platform. Hence, an external device cannot directly be attached. The task of a proxy is to create a dedicated network link for connecting to the particular KNX network the physical device is attached to.

This functionality opens up several possibilities for enhancing interaction and usage of the learning platform. It allows testing of devices, i.e., by selectively replacing a device modeled in the platform with its physical counterpart. Another use-case is creating joint systems via combination of platform setups and physical installations.

4.5 Use-case: Automating an Apartment

The prototype of the learning platform should provide an easy-to-understand, but yet versatile scenario in which automation principles can be applied. Such scenario should cover several possibilities of designing the communication network, choose from various sensors and actuators to connect to the network, and visualize the differences of executed communication protocols.

The selected scenario to provide such a use-case is an apartment of a domestic building. The apartment is subject to function as *smart home*. For that purpose, the appliances in the apartment are considered *smart*, and allow communication via KNX. The advantage of the apartment use-case is that also younger students participating in children's university activities already possess a conceptual understanding of common appliances in a household. On the other hand, the field of domotics covers a wide range of applications with complex control tasks suited for graduate student laboratory work.

The main area of the GUI is determined by the building apartment floor plan. That floor plan is provided as resource and loaded during initialization by the platform's view component. The GUI also offers usable appliances as a collection of supported devices with KNX functionality. After selecting a device, the user places it onto the floor plan. The permitted locations depend on the specific device type; in general, the spatial placement is within apartment rooms, or at the exterior building walls.

Once placed, the devices enable their sensor or actuator functionality.

4.5.1 Device Configuration

Device configuration is possible via three modes, which are not exclusive but complement each other.

Automatic Mode In automatic mode, a newly inserted device is automatically configured by the platform. That way, a user gets an immediate working device.

Manual Mode In manual mode, the user has the ability to directly modify selected settings of a device using the GUI. The most interesting settings that are accessible using the GUI are the assigned datapoints for process communication, and the device's individual address.

Programming Mode Each device offers a programming button. That button corresponds to the KNX programming button, which puts a device into programming mode. In programming mode, the user can execute a sequence of specified KNX device management procedures for further configuration.

4.5.2 Process Communication

The foremost application of the learning platform involves the setup of, and patterns for, KNX process communication. Main user tasks are creating, observing, and adapting group communication patterns, and evaluating the effects on the target devices. Establishing the logic for operation involves querying sensor values, and sending commands to actuators.

Entity	*Media/Protocols*
Gateway (server-side)	KNXnet/IP Tunneling & Routing, KNX IP
KNX subnet	KNX IP[1], TP1[2], PL110[2], RF[2]

Table 4.2: Platform support of KNX communication media and protocols.

Where feasible and intuitive for a user, sensors might allow direct interaction via the GUI to alter state or trigger events, e.g., pressing a push-button control on a device which represents a light switch in a room. For devices implementing sensor functionality, of primary interest is the inspection or interaction with datapoint values of the state- or command-based datapoints assigned to a particular device.

For manual interaction, easy accessibility in the GUI to datapoint values shall support the user. A non-intrusive way of presenting those values is by using balloon tip components as described in section 3.3.2 on page 28.

Junior Mode

In junior mode, device configuration via programming mode is not enabled by default. The main reason is the required advanced knowledge of KNX configuration procedures.

Graduate Mode

Besides mapping sensor output directly to actuator operations via datapoints, graduate mode provides a controller component. The controller allows to create simple expressions on datapoint values, e.g., compare two inputs and perform a conditional operation depending on the result. This functionality enables more comprehensive interaction and extends the logic for operation in process communication.

4.5.3 Summary

This chapter described the model and operation of the learning platform. The overall model consists of the communication model, device model, weather model, and energy model. From the perspective of operation, the crucial part is the communication model, being the foundation of the overall platform design and key to running the learning platform. The model supports several communication media and protocols; table 4.2 shows an overview of the available options for access and communication.

1 Using the host's Ethernet interface or the virtual loopback interface.
2 Virtual link.

The use-case for the prototype features the versatile scenario of automating an apartment. Figure 4.7 shows a screenshot of the main window of the learning platform software prototype after startup.

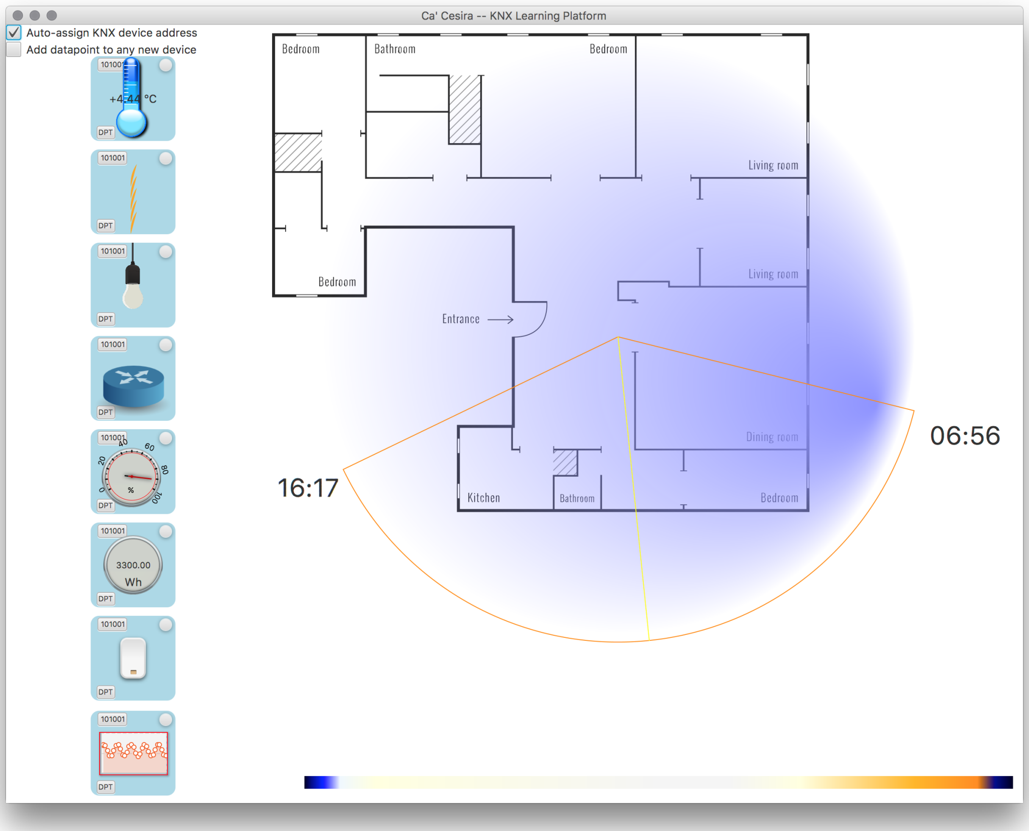


Figure 4.7: Screenshot of the learning platform's main window after startup. On the left side is the device bar with sensor and actuator devices. The window center shows the apartment layout, in this example a floorplan loaded from a Scalable Vector Graphics (SVG) file. At the bottom is a gradient of the sky color, used for testing purposes.

CHAPTER 5

Conclusion & Outlook

The study of this thesis is founded in the movement towards more software-based solutions in the field of domotics. This development makes it possible to approach a learning platform that does not require the integration of, or immediate access to, laboratory equipment. Still, the learning effect is kept due to the use of the same protocol stacks as deployed in a physical setup.

5.1 Summary

This work proposed and examined an interactive learning approach for domotics using a software-based learning platform. The results of the work provide an alternative for students to high investment costs in obtaining laboratory equipment, and limited availability of resources in the laboratory. A consequence of those limiting factors often results in the necessity for students to adhere to strict schedules at the university for laboratory course work. Another contribution of this work is to support distant learning, by supporting software distribution to user computer platforms.

Of particular interest for the platform design were two user groups. On the one hand young students approaching middle school, on the other hand graduate level university students. The intention is to complement introductory courses teaching control and automation theory and introducing the concepts of communication protocols, as well as laboratory course work.

The theoretical part of the thesis was concerned with providing the foundation for a learning platform design, to accommodate the needs of the two target groups. A main design focus was to feature communications and applications in building automation. To target younger students, the platform's *junior* mode follows a serious game concept. A user can choose among quests, e.g., energy-efficient living, or preparing an apartment for a storm. *Graduate* mode is for graduate university students, with the focus on creating automation setups, in order to understand and execute laboratory assignments. The design in graduate mode supports the predominant tasks in control and automation.

After discussing the different choices of programming languages, the decision was made in favor of Java. For development, various Java UI toolkits and frameworks were examined. Priority was given to options of free and open source software, and libraries that ship as part of the programming language. This simplifies sharing, distribution and modification, also for future students.

For the practical part, a prototype was developed in Java and JavaFX, based on the proposed design. The selected use-case is an apartment serving as basis for a *smart* home. The resulting platform features a device model with a selection of KNX sensors, actuators, and controllers, a communication model supporting the KNX communication media, a weather model, and an energy model for electricity. Development focused on creating cross-platform software, to also allow deployment to smart devices using the operating systems Android or iOS. Appendix A on page 52 provides a summary of the requirements, build system, and software dependencies of the developed software.

5.2 Future Work

Further studies are necessary for the link abstraction over particular communication media. Depending on the communication medium, a link should reflect a medium's characteristics by maintaining its most important properties. In the developed prototype, only KNXnet/IP and KNX IP can operate close to a physical setup by communicating over the local Ethernet or virtual loopback interface. But for TP1, PL110, or RF, the link currently implements a medium-agnostic abstraction. Future work could specify a more accurate physical layer model of the TP1 communication medium, currently being used in most KNX installations.

A short discussion in section 4.4.3 explained the possibilities of using the learning platform in combination with physical devices. This requires only a proxy device in the platform, with the exception of KNX IP where not even a proxy is necessary. Evaluating in depth the use-cases where that functionality proves useful, and discussing possible synergies can provide further insight for moving towards such mixed laboratory setups.

Regarding the energy model, one enhancement could be a controller containing building-logic with more complex control strategies for an energy-efficient home. For example, study scheduling problems like load management schemes for flattening electricity usage.

Several opportunities for enhancement also poses the integration of device-specific profiles. The platform in its current prototype stage supports offline configuration of device types. But to create a detailed device profile, the configuration setup requires a user to have domain-specific knowledge. Here, existing literature in the area of smart grids and smart living have different attempts to describe profiles of household appliances. Profile types range from usage-profiles to profiles of power consumption, as well as profiles that represent function blocks used by different device categories. A method to integrate such profiles would improve representation of devices in the learning platform.

APPENDIX A

Prototype Development

A short overview of requirements, build system, and software dependencies of the developed software prototype.

A.1 Build Requirements

The prototype was developed with source compatibility and minimum Java runtime requirements for Java 8 SE. The graphical components are based on the JavaFX 8 platform. As a build automation system, Gradle was used. To enable cross-platform deployment of JavaFX applications on mobile platforms, the gradle plugin `javafxmobile-plugin` [55] was used, with version 1.0.6. The plugin unifies builds for Desktop, Android, and iOS platforms.

The Calimero network stack version implemented in the prototype for KNX communication is version 2.3 for Java 8. JSON parsing for the weather model data is performed with Gson [56], version 2.4. For parsing mathematical expressions provided in the configurations of the energy model, the prototype depends on `exp4j`, version 0.4.5. To support differential loads in the energy model, Apache `commons-math3` is used, version 3.5.

A.2 Test Platforms

As main test platforms served OS X 10.11.1 (El Capitan) and Ubuntu 15.10 with Unity. The installed Java Development Kit (JDK) was Oracle Java SE (Mac OS X x64 and Linux x64), version 1.8.0_40.

A.3 Logging

The learning platform uses the Simple Logging Facade for Java (slf4j). Its facade architecture allows to bind any desired logging framework. By default, the learning platform assumes the slf4j *Simple Logger* binding. The simple logger logs everything to standard output, and can be configured via the `simplelogger.properties` file, Java Virtual Machine (JVM) system properties, or Java command-line options.

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 15, 16, 21, 25, 38–40

ASCII American Standard Code for Information Interchange. 19

AWT Abstract Window Toolkit. 31

BACNET Building Automation and Control Networks. 12

BATIBUS BatiBUS. 12

BCU Bus Coupling Unit. 15, 42

BSD Berkeley Software Distribution. 15

CDC Connected Device Configuration. 16

CEMI common External Message Interface. 13, 19, 42

CEN Comité Européen de Normalisation. 12

CENELEC Comité Européen de Normalisation Électrotechnique, European Committee for Electrotechnical Standardization. 12

CSCE Computer-Supported Collaborative Environment. 8

CSS Cascading Style Sheets. 30, 32

DCOM Distributed Component Object Model. 16

DPT Datapoint Type. 13, 15, 16, 18, 19

EHS European Home Systems Protocol. 12

EIB European Installation Bus. 12, 15, 16

EMI External Message Interface. 17, 42

EMI1 External Message Interface 1. 13, 19

EMI2 External Message Interface 2. 13, 19

EN Europäischen Normen. 12

EPL Eclipse Public License. 31

ETS Engineering Tool Software. 14, 15

FP Foundation Profile. 16

FT1.2 FT1.2. 15, 17, 18

FXML FXML. 31

GB/T Guobiao, Voluntary National Standards in China. 12

GPL GNU General Public License. 15, 16

GUI graphical user interface. 4, 6, 8, 25–29, 31, 32, 34, 35, 41, 43, 45, 46

GWT Google Web Toolkit. 30

HTML HyperText Markup Language. 29, 39

HTML5 HyperText Markup Language Revision 5. 30

HVAC Heating, Ventilating, and Air Conditioning. 4, 9, 40

ID Identifier. 39

IDE Integrated Development Environment. 32

IEC International Electrotechnical Commission. 9, 12

IEEE Institute of Electrical and Electronics Engineers. 11

IOT Internet of Things. 1

IP Internet Protocol. 13, 15–18, 43, 49

IR Infrared. 13

ISO International Organization for Standardization. 9, 12, 19

JDK Java Development Kit. 51

JSON JavaScript Object Notation. 39, 51

JVM Java Virtual Machine. 52

KNX KNX. 4, 6, 7, 9, 11–20, 24, 26, 27, 29, 33, 34, 36, 42–46, 49, 51

KNX IP KNX communication over IP. 13, 18, 42, 43, 46, 49

KNXNET/IP KNXnet/IP. 6, 13, 15–18, 43, 44, 46, 49

LGPL Lesser GNU General Public License. 15

ME Micro Edition. 16, 31

MIT Massachusetts Institute of Technology. 15

MOC Meta-Object Compiler. 32

MVA model–view–adapter. 26

MVC model–view–controller. 25, 26, 29, 31, 34

OS Operating System. 32

OSI Open Systems Interconnection. 12, 25

PAN personal area network. 11

PEI Physical External Interface. 15

PL110 Power-line 110. 12, 13, 18, 42, 44, 46, 49

PL132 Power-line 132. 12, 13, 18

PLAF pluggable look and feel. 31

QML Qt Meta-Object Language. 32

R&D research and development. 2

RCP Rich Client Platform. 31, 33

REST Representational State Transfer. 38, 39

RF Radio Frequency. 13, 17, 18, 42, 44, 46, 49

RIA Rich Internet Application. 32

RPM RPM Package Manager. 15

SDK software development kit. 15

SE Standard Edition. 16, 32, 51

SLF4J Simple Logging Facade for Java. 52

SNVT Standard Network Variable Type. 12

SOC System-on-Chip. 1

SQL Structured Query Language. 38

SVG Scalable Vector Graphics. 47

SWT Standard Widget Toolkit. 31, 32

TP0 Twisted Pair 0. 12, 13

TP1 Twisted Pair 1. 12, 13, 17, 18, 42, 44, 46, 49

TP-UART Twisted-Pair Universal Asynchronous Receiver Transmitter. 15, 17

UDP User Datagram Protocol. 43

UI user interface. 21, 29–33, 49

USB Universal Serial Bus. 17, 18

VSM Virtual System Modelling. 8

WOEID Where On Earth ID. 38

XML Extensible Markup Language. 31, 39

YQL Yahoo Query Language. 38, 39

ZIP Zone Improvement Plan (postal code). 39

Bibliography

[1] KNX Association. `www.knx.org`.

[2] Institute of Computer Aided Automation. `https://www.auto.tuwien.ac.at/a-lab/research.html`.

[3] Jing Ma and Jeffrey V. Nickerson. Hands-on, Simulated, and Remote Laboratories: A Comparative Literature Review. *ACM Computing Surveys*, 38(3), 2006.

[4] C.E. Pereira, S. Paladini, and F.M. Schaf. Control and automation engineering education: Combining physical, remote and virtual labs. In *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, pages 1–10.

[5] LEGO Mindstorms. `mindstorms.lego.com`.

[6] Proteus VSM. `www.labcenter.com/products/vsm/vsm_educational.cfm`.

[7] P. Brejcha, R. Beneder, and M. Kramer. New approaches for a distance learning course about embedded systems. In *Global Engineering Education Conference (EDUCON), 2011 IEEE*, pages 903–906.

[8] E.A. Mossin, R.P. Pantoni, D. Brandao, and N.M. Torrisi. Fieldbus simulator: Architecture, typical experiment and tool evaluation. In *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, pages 3512–3517.

[9] J. Bravo and M. Ortega. Planning in problem solving: a case study in domotics. In *Frontiers in Education Conference, 2000. FIE 2000. 30th Annual*, volume 1, pages T2D/11–T2D/16 vol.1.

[10] M. Latorre Garcia, G. Carro Fernandez, E. Sancristobal Ruiz, A. Pesquera Martin, and M. Castro Gil. Rethinking remote laboratories: Widgets and smart devices. In *Frontiers in Education Conference, 2013 IEEE*, pages 782–788.

[11] Siemens. GAMMA Training Kit 4 V 3 (GTK4.3), 2011.

[12] Peter Fritzson. *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley-IEEE Press, September 2011.

[13] Dassault Systèmes AB. Dymola – Dynamic Modeling Laboratory. `http://www.3ds.com/products-services/catia/products/dymola`.

[14] *Information technology – Home Electronic Systems (HES) – Part 3-10: Wireless Short-Packet (WSP) protocol optimized for energy harvesting – Architecture and lower layer protocols*. ISO/IEC 14543-3-10:2012.

[15] EnOcean Alliance Inc. System Specification – Executive Summary. Generic Profiles v1.0, June 2013. San Ramon, CA, USA.

[16] IEEE. *Local and metropolitan area networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Std 802.15.4-2011.

[17] *Building automation and control systems (BACS) – Part 5:Data communication protocol*. ISO 16484-5:2014.

[18] ANSI/ASHRAE. *BACnet – A Data Communication Protocol for Building Automation and Control Networks. Addendum q*. ANSI/ASHRAE 135-2008q.

[19] *Information technology – Control network protocol – Part 1: Protocol stack*. ISO/IEC 14908-1:2012.

[20] *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. ISO/IEC 7498-1:1994.

[21] *Volume 3: System Specifications – Part 1: Architecture – Chapter 1: Architecture*. Konnex Association, 2004. v2.1.

[22] KNX. *Volume 3: System Specifications – Part 3: Communication – Chapter 2: Data Link Layer General*. v1.0.

[23] KNX. 3/2/5: System Specifications – Communication Media – Radio Frequency, 2008. Version 1.0.00.

[24] KNX. 3/2/6: System Specifications – Communication Media – KNX IP, 2010. Version 1.0.00.

[25] *System Specifications – Interworking – Interworking Model*. 2009. KNX 03/07/01 – v1.4.

[26] KNX Association. ETS5 Flyer. ETS5: one tool for ALL media. `http://www.knx.org/media/docs/Flyers/ETS5-Flyer/ETS5-Flyer_en.pdf`.

[27] Dietmar Dietrich, Wolfgang Kastner, and Thilo Sauter. *EIB: Gebäudebussystem*. Hüthig Verlag, Heidelberg, 2000.

[28] Life Emotions. KNX.net. `http://lifeemotions.github.io/knx.net`.

[29] Martin Kögler. *Free Development Environment for Bus Coupling Units of the European Installation Bus*, March 2011.

[30] knxd. knxd on GitHub. `https://github.com/knxd/knxd`.

[31] KNX User Forum. eibd(war bcusdk) Fork -> knxd. `http://knx-user-forum.de/forum/%C3%B6ffentlicher-bereich/knx-eib-forum/39972-eibd-war-bcusdk-fork-knxd`.

[32] Frédéric Mantegazza. pKNyX project. `http://www.pknyx.org`.

[33] Bernhard Erb, Wolfgang Kastner Georg Neugschwandtner, and Martin Kögler. Open-source foundations for PC based KNX/EIB access and management. In *KNX Scientific Conference, 2005*.

[34] Calimero Project. Java libraries for KNX access and management. `https://github.com/calimero-project`.

[35] SourceForge. Calimero Project File Hosting. `https://sourceforge.net/projects/calimero`.

[36] Calimero Project. Calimero distribution v2.1. `www.calimero.sourceforge.net`.

[37] Oracle Java SE 8. Compact Profiles. `https://docs.oracle.com/javase/8/docs/technotes/guides/compactprofiles/compactprofiles.html`.

[38] Gartner, Inc. Press Release: Gartner Says Replacement Indecision Will Cause Device Shipments to Decline 1 Percent in 2015, September 2015. Egham, UK. `http://www.gartner.com/newsroom/id/3135618`.

[39] Dean Wampler and Alex Payne. *Programming Scala*. O'Reilly, Sebastopol, CA, USA, 2nd edition, November 2014.

[40] W3C. HTML5 – A vocabulary and associated APIs for HTML and XHTML – W3C Recommendation 28 October 2014. `http://www.w3.org/TR/html5/single-page.html`.

[41] ECMA-262, ECMAScript 2015 Language Specification, June 2015. 6th Edition.

[42] W3C. CSS Snapshot 2015 – W3C Working Group Note, 13 October 2015. `http://www.w3.org/TR/css-2015`.

[43] Google Inc. Google Web Toolkit. http://www.gwtproject.org.

[44] jUnit. Unit Testing Framework. http://junit.org.

[45] Oracle. Java Platform, Standard Edition 7 API Specification – Package java.awt. http://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html.

[46] Oracle. Java Platform, Standard Edition 7 API Specification – Package javax.swing. http://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html.

[47] SwingLabs. SwingX. https://swingx.java.net.

[48] Steve Northover and Mike Wilson. *SWT: The Standard Widget Toolkit, Volume 1*. Addison-Wesley Professional, 1st edition, 2004.

[49] Jasper Potts, Nancy Hildebrandt, Joni Gordon, and Cindy Castillo. *JavaFX – Getting Started with JavaFX, Release 8*. Oracle, August 2014. E50607-02.

[50] Apache Software Foundation. Apache Pivot. https://pivot.apache.org/index.html. version 2.0.4.

[51] Jürgen Bocklage-Ryannel and Johan Thelin. *Qt5 Cadaques*. http://qmlbook.github.io, July 2015. Release 2015-03.

[52] Qt Jambi – Qt for Java. http://qtjambi.org.

[53] Yahoo Developer Network. Yahoo Weather API. https://developer.yahoo.com/weather.

[54] OpenWeatherMap. Weather API. http://openweathermap.org/api.

[55] JavaFXPorts. javafxmobile-plugin gradle plugin. https://bitbucket.org/javafxports/javafxmobile-plugin.

[56] Inderjeet Singh, Joel Leitch, and Jesse Wilson. Google Gson User Guide. https://sites.google.com/site/gson/gson-user-guide.