Diss. ETH No. 18876

Mobile Devices for Interacting with Tagged Objects: Development Support and Usability

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
Christof Roduner
M.Sc. in Computer Science, University of Zurich
born December 28, 1978
citizen of Sennwald SG, Switzerland

accepted on the recommendation of
Prof. Dr. Friedemann Mattern, examiner
Prof. Dr. Elgar Fleisch, co-examiner
Prof. Dr. Marc Langheinrich, co-examiner

2010

Abstract

Augmenting everyday physical objects with digital services is one of the main interests of research in ubiquitous computing. In the pursuit of this goal, personal mobile devices, and mobile phones in particular, play an important role. As a powerful, networked, and highly personal computing platform that is always carried along by most people, mobile phones represent an ideal tool to mediate the interaction between users and an environment that is enriched with ubiquitous computing technology. They have thus become the default physical user interface for many ubiquitous computing applications.

More recently, mobile phones have been equipped with technologies that allow them to be used as reading devices for inexpensive passive identification tags. These technologies include Near Field Communication (NFC) for the detection of RFID labels as well as the simple mobile phone camera, which can be leveraged to recognize standard barcodes as can be found on the majority of consumer goods.

As a result of this, it is now possible to augment virtually all physical products with digital services without the need to deploy additional hardware. This raises two important challenges. First, while a wide range of possible applications and benefits for users have been proposed by the research community, it remains unclear which of these can live up to the promise of making our daily lives simpler, safer, or more efficient. Second, the sheer number of everyday objects that can potentially be augmented with digital services calls for the ability to create such services in a fast and efficient way that goes beyond the ad-hoc manner in which such applications are typically developed today.

This thesis addresses these challenges with three main contributions. First, it investigates the specific case of using mobile phones to interact with physical appliances. It provides a classification of interaction task types and demonstrates in a user study which types are suitable for phone-mediated interaction and which are not. Second, it proposes an architecture and implementation of an infrastructure to publish and discover services for tagged objects that frees service providers from the need to develop their own custom solutions and allows users to find relevant services based on their context. Third and last, it presents the design and implementation of a software framework that significantly simplifies the development of digital services for tagged objects on mobile phone platforms.

Taken together, these contributions facilitate the rapid creation of usable digital services for tagged everyday objects and lay the foundations for a wider adoption of innovative applications in this area.

Kurzfassung

Eines der Hauptinteressen des Ubiquitous Computing ist es, physischen Alltagsgegenständen eine zusätzliche Dimension zu geben, indem sie mit digitalen Diensten angereichert werden. Dabei spielen mobile Geräte und insbesondere Mobiltelefone eine wichtige Rolle. Dank ihrer Vernetzung, ihrer einfachen Programmierbarkeit und ihres persönlichen Charakters sind sie ein idealer Vermittler in der Interaktion zwischen Benutzern und einer Umwelt, die auf vielfältige Art und Weise mit Ubiquitous-Computing-Technologien ausgestattet ist. In vielen Anwendungen sind Mobiltelefone daher zur eigentlichen physischen Benutzungsschnittstelle des Systems geworden.

Neuere Mobiltelefone sind zunehmend auch mit Technologien ausgerüstet, die eine kostengünstige Identifikation von physischen Gegenständen erlauben. Beispiele dafür sind die Near Field Communication (NFC) oder auch nur die eingebaute Kamera, womit passive RFID-Tags erkannt bzw. die weit verbreiteten Produkt-Barcodes ausgelesen werden können.

Dies ermöglicht es, ohne grossen Aufwand nahezu alle physischen Produkte mit digitalen Services zu erweitern. Damit verbunden sind jedoch zwei zentrale Herausforderungen: Erstens ist nicht klar, wo der Einsatz solcher Dienste aus Benutzersicht tatsächlich Sinn ergibt. Zwar wird gegenwärtig eine Vielzahl von Anwendungen diskutiert, die aus technischer Sicht interessant sind. Jedoch stellt sich die Frage, welche davon das Versprechen einer Vereinfachung unserer alltäglichen Lebenswelt auch tatsächlich einlösen. Zweitens entsteht mit der grossen Zahl von Gegenständen, für die solche Erweiterungen in Frage kommen, die Notwendigkeit, die digitalen Services rascher und effizienter zu entwicklen, als dies mit den heute gebräuchlichen Ad-hoc-Ansätzen der Fall ist.

Die vorliegende Dissertation liefert drei Hauptbeiträge zu diesen Herausforderungen: In einem ersten Teil wird der Einsatz von Mobiltelefonen zur Bedienung alltäglicher Geräte untersucht und eine Klassifikation von Interaktionsarten vorgestellt. In einer Benutzerstudie wird daraufhin gezeigt, für welche davon die Unterstützung durch ein Mobiltelefon zu einer verbesserten Benutzbarkeit führt und für welche dies nicht der Fall ist. In einem zweiten Teil stellt die Dissertation die Architektur und Implementierung einer Infrastruktur vor, mit der Dienste mit Alltagsgegenständen verknüpft werden können. Dies erlaubt es Benutzern, die für sie relevanten Dienste unter Berücksichtigung des jeweiligen Kontexts zu finden. Schliesslich liefert der dritte Teil der Arbeit ein Software-Framework für Mobiltelefone, welches auf der im zweiten Teil vorgestellten Infrastruktur aufbaut und die Entwicklung von digitalen Services für Alltagsgegenstände deutlich erleichtert.

Insgesamt ermöglichen diese Beiträge die raschere und einfachere Einführung von benutzbaren Diensten für Alltagsgegenstände und schaffen somit eine der Voraussetzungen für die Umsetzung innovativer Anwendungen im Gebiet des Ubiquitous Computing.

Contents

1. Introduction

1.1. Background

Personal mobile devices, such as PDAs and mobile phones, have long represented an important building block of many systems discussed in the ubiquitous computing community [26, 54, 63, 72, 99]. Today, even relatively simple mobile phones have many characteristics that make them appear as an ideal tool for the implementation of novel ubiquitous computing applications. As a cheap general purpose computing platform that is equipped with input / output, storage, and communication capabilities, mobile devices are also relatively easy to program. Many of the models available today also ship with built-in sensor modules in the form of accelerometers, embedded cameras, or GPS receivers, which greatly facilitates the creation of context-aware applications [29]. Replacing palmtop and tablet PCs used in early context-aware systems [1, 22], they have often become the platform of choice for such applications today. In addition, the combination of these features allows for new interaction techniques that, together with their prevalence, make mobile phones the default physical interface for ubiquitous computing applications [8]. From a user perspective, they are, due to their personal nature, highly familiar, trusted, and always in reach. Researchers in the ubiquitous computing domain have thus used them for prototypical implementations of a wide range of applications as diverse as attaching digital annotations to physical objects [124], locating lost personal items [45], interacting with large public displays [122], controlling a pointing device remotely [85], or operating appliances of all sorts [98].

Another prominent theme in ubiquitous computing is auto-id technology, i.e., the automatic identification of physical objects. Its most widely known form is the omnipresent EAN/UPC product barcode, which has been used extensively since as early as the mid-1970s, when barcode labels began to speed up the checkout process in retail stores. Today, barcodes have become truly ubiquitous, forming the backbone of many automated processes, such as in airline ticketing and baggage handling, in libraries and video rental shops, in hospitals, and — most of all — in industrial supply chain management.

During the past few years, radio frequency identification (RFID) technology has received considerable attention, as it has the potential to significantly improve today's supply chain [42]. RFID technology can enable cost savings because it offers two distinct advantages over barcodes: Firstly, RFID labels do not require a line of sight between a reader and a tag, thereby allowing large numbers of tags to be read quickly and without manual labor. Secondly, traditional one-dimensional EAN/UPC barcodes can only be used to identify products at a class level, while RFID tags allow for the identification of individual items, thus enabling more detailed product tracking capabilities.

Both technologies, barcodes as well as RFID tags, have been used in ubiquitous computing as an inexpensive means of bridging the gap between the physical world with tangible items and the virtual world offering information and services related to these items. The idea of linking information and services to physical objects has been investigated in numerous research projects, using both printed markers [65, 84, 116, 123, 137] as well as less obtrusive embedded RFID tags [145, 76, 126, 23].

More recently, the two fields of personal mobile devices and auto-id technology have moved closer together with the advent of technologies that allow mobile phones to read both visual as well as radio frequency tags. For example, most phones feature an integrated camera, which has been shown to be capable of decoding the omnipresent EAN/UPC barcode symbols [3]. On top of this, some mobile phones are already equipped with a Near Field Communication (NFC) module [102], which is able to read passive RFID tags.

The convergence of tagging technologies and mobile phones makes it possible for users to interact with products through mobile devices. As the large body of ubiquitous computing research in this field illustrates, the paradigm per se is not novel. Yet what is new is the scale at which it can be applied. As the underlying technology — mobile phones with built-in cameras or NFC modules, barcodes, and RFID tags — is fundamentally inexpensive and readily available, it is no longer just dedicated objects that can offer digital services in the form of a "virtual counterpart" [126]. It is now possible for almost *every* ordinary product to be augmented with digital services. As long as it is labeled with a barcode or RFID tag, it can serve as a starting point for mobile phone-based interaction.

1.2. Opportunities

From an industry perspective, this development opens many very attractive opportunities for businesses, but it also poses challenges. While the benefits of auto-id tags were earlier limited to internal business processes (e.g., enhanced efficiency in supply chain management), it is now possible to leverage this technology throughout a product's life cycle.

One of the most promising prospects is that businesses are now able to tie digital offerings to any physical product in an intuitive way. Consumers can access these digital services by interacting with the physical product and simply following the “physical hyperlinks” that are provided. In effect, this allows a tangible product's intangible dimension [81] to be enriched and extended in novel ways. While an item's packaging is inherently limited in area, the mobile device offers another channel for businesses to provide additional product intangibles.

These developments have a number of implications for product manufacturers, third-party businesses, and customers alike:

- For a *manufacturer*, it is now possible to deliver added value to customers by enriching its physical products with digital services that can be accessed in a straightforward and intuitive way. Unlike the physical product itself, such digital services are not static and can evolve over time, thereby ensuring an ongoing appeal and repeated interaction with the product. If these services are tailored to a consumer's needs, they also allow for the personalization of a product, which would not be feasible with the physical item alone. Finally, repeated interaction between a consumer and a product can be a valuable source of information. By analyzing these interaction patterns, a manufacturer can gain a better understanding of a product's actual use and learn about a customer's preferences. Such insight can be used both to further develop the relationship with a customer, and to improve a product to better meet consumer needs. Overall, a manufacturer can leverage the combination of tagged products and mobile devices to differentiate its offering and improve customer loyalty.

- Opportunities for services based on tagged products are not limited to manufacturers, but arise for *third-party businesses* as well. For a consumer advocacy organization, for example, it is very attractive to "link" a review directly to a physical product, thus making it easily accessible when a buying decision is made in a brick and mortar store. Similar benefits can be reaped by other businesses that currently offer product-related information on the internet. A price comparison service, for example, can greatly benefit from the fact that users can access its data directly at the point of sale and without manually typing the product name into a search form.

- For *customers*, these developments lead to a host of new services for physical products, with their main benefits lying in easier accessibility, higher transparency, and more convenience. For example, users could now use their mobile phones to control electrical appliances instead of relying on the sometimes confusing haptic user interface of devices, such as vending machines, coffee makers, or washing machines. Since user interfaces on the mobile phone can easily be tailored to users' specific preferences and needs (such as their native language or output for the visually impaired), they are assumed to simplify the interaction with appliances. Better accessibility is, however, not limited to appliance control. In a shopping environment, for example, consumers can now easily retrieve detailed reviews and other background information on a product without the cumbersome step of typing its name into a search form. While reviews have been available on the internet for quite some time, a considerable effort was required in order to access them in a store, i.e., at the moment when they are most useful. Ultimately, easier accessibility of such services can lead to more transparency, since consumers will likely have better knowledge of cheaper offerings, unhealthy product ingredients, or questionable manufacturing practices. Finally, products can become more convenient to use thanks to the digital services they provide. For example, it should no longer be needed for a customer to keep paper copies of certificates of warranty, since the electronic version is accessible by simple touching the RFID tag of a defective product. A possible "digital problem solving service" could even go a step further and read an appliance's error code, which is used together with the phone's GPS sensor values to locate the nearest repair center.

In summary, the paradigm of digitally augmenting physical objects has the potential to transform business around physical products in several ways. It allows companies to improve their customer relationships by offering new channels for accessibility, new ways for achieving awareness, and new techniques for responding [38]. Applying these principles can help pure product vendors make their offering "smart" and position themselves as service providers [5]. In addition, tagging technologies pave the way for mobile "interactive decision aids" that can simplify the shopping experience, which in the eyes of many has become too complex due to the myriad of alternative offerings available today [57]. What starts out as a way for some businesses to differentiate themselves could become an inevitable reality for others, as products that are unable to actively provide information outside the store could soon be perceived as "dumb" and unattractive [40].

1.3. Motivation

While these seem to be promising opportunities, the reality for those who consider to implement such offerings is less clear-cut today. There is no conclusive evidence that mobile phones with their small form factor and still fairly rigid interaction possibilities are in fact suitable tools for all the scenarios that are being discussed. For example, it remains unclear whether they can serve as an adequate replacement for some of the traditional haptic user interfaces that we use in everyday life today. While many of the scenarios that are often cited in the research community sound appealing, it is disputable whether they can stand the proof of broad and regular use. Apart from these open issues in the field of human-computer interaction, there are a number of purely technical problems that hinder the adoption of services based on tagged objects:

Lack of abstraction The programming of mobile devices has become considerably simpler in the past few years. However, in the light of software development techniques that are widely used in web programming, and that have kicked off a wave of innovative new applications in the internet world, mobile phone programming still has a long way to go. Creating services for the mobile phone is often a time-consuming, tedious, and error-prone task. A lack of abstraction forces developers to deal with the intricacies of the underlying platform (e.g., its specific GUI framework), the various communication technologies that are available (e.g., GPRS, WiFi, Bluetooth, or NFC), as well as specific sensor APIs (e.g., for auto-id tags, location, or acceleration). The current situation essentially requires service providers to develop a bespoke implementation for every single target platform, such as Nokia's Symbian S60, Google's Android, Microsoft's Windows Mobile, or Apple's iPhone OS. This incurs longer development cycles, higher costs, and limited agility.

Monolithic applications Mobile phone applications are mostly monolithic today. Much like in the traditional desktop computing world, they are self-contained packages that need to be downloaded, installed, and updated manually. As a result of this, users have to *start an application* when they actually wish to *access a service* that is offered by a tagged product. While this may initially appear to be a minor inconvenience for users, it constitutes a rather fundamental problem as the number of available services grows. At a certain point, users will already have a sizable number of applications installed and will be reluctant to consider any new services. This prevents the whole idea of digitally augmenting tagged products from scaling.

Lack of infrastructure There is currently no backend infrastructure that allows for the discovery and exchange of information and services pertaining to tagged objects. Existing solutions are rooted in the logistics industry and closed by design. Consequently, every service provider will need to define its own data format to structure, describe, and exchange product-related information and services. Not only does this come at a significant cost, but the lack of support for publishing and discovering product-related services also leads to users not finding the resources they look for.

1.4. Objective

As outlined in the previous section, there are a number of specific obstacles for augmenting tagged objects with digital services. The goal of this thesis is to identify factors that help overcome these barriers in order to leverage the potential of phone-based interaction with tagged objects. Our approach is twofold: On the one hand, we seek to answer the question of *when* it makes sense to use the mobile phone as an interaction device and when it does not. In particular, we investigate the degree to which interaction with physical objects can be shifted from traditional, real-world user interfaces to virtual user interfaces on the mobile phone. On the other hand, we analyze *how* the creation of such services for tagged objects can be facilitated. More precisely, we propose both infrastructure components and a software framework that address the challenges presented above by enabling the rapid and lightweight development and deployment of mobile phone-based services for tagged objects.

We pursue a dual approach that comprises both a human-computer interaction perspective as well as an infrastructure perspective because both represent crucial factors for driving adoption today. A solution will not be successful unless it is usable by mobile phone owners, and it will not be offered unless it is economical for implementers.

1.5. Contributions

In the previous sections, we highlighted some of the challenges that arise for those who consider augmenting tagged objects with digital services on the mobile phone. In this section, we present the main contributions that are made by this thesis in order to address these challenges:

1.5.1. Usability evaluation

We investigate the specific case of using mobile phones to operate physical everyday electrical appliances, such as, for example, printers, coffee makers, or washing machines. Traditional interaction based on haptic user interfaces is compared with phone-mediated interaction. A user study is carried out to test a set of hypotheses regarding the value of mobile phones in this area. In contrast to similar work that is based on paper prototypes, our evaluation uses actual physical appliances and mobile phones. In particular, our evaluation shows that:

- in situations that are not encountered on a daily basis, offloading the user interface from the physical appliance onto the mobile phone results in faster and less error-prone interaction;

- in such situations, phone-based user interfaces are preferred by users over traditional haptic user interfaces;
- troubleshooting an appliance in an erroneous state is faster when it is assisted by a mobile phone instead of a paper-based user manual;
- mobile phones do not represent suitable interaction devices for carrying out everyday tasks with an appliance.

1.5.2. Infrastructure for publishing and discovering information and services for tagged objects

This thesis also provides an *open lookup infrastructure* that allows any interested party to publish and advertise information and services for tagged products. The open lookup infrastructure offers a simple way to describe such information and services and to integrate third-party data. It supports the retrieval of relevant information based on the user's context. Its design is "open" in the sense that it is not centralized and possibly under the control of a single entity. Instead, any stakeholder can link its own information and services to any tagged object available.

A prototypical implementation of the open lookup infrastructure is presented to validate its concepts. Through several demonstrators that we built on top of it, we show that the open lookup infrastructure is well suited to facilitate the development of applications around tagged products.

1.5.3. Concepts and framework for the development of mobile services for tagged objects

As a third contribution, this thesis provides a framework that facilitates the development of services for tagged objects that run on a mobile device. This framework offers a number of abstractions that free developers from dealing with many aspects of applications that are usually not a main concern in services for tagged objects. It is lightweight and supports the on-the-fly discovery and installation of new services. Comparing to the current state of the art in software development for mobile phones, our framework offers a considerably faster development process, is easier to learn by developers, and allows for the creation of services that are portable among different mobile phone platforms.

This thesis also presents BIT, a *browser for the Internet of Things.* BIT is a browser application that runs on a user's mobile device and provides a prototypical implementation of our framework. From a user's perspective, BIT is a "single point of interaction" that offers unified access to all information and services that are available for a tagged object at hand. In order to do so, it leverages the second contribution, the infrastructure for publishing and discovering information and services. We illustrate the value of our framework by implementing several example services of different types. By comparing a traditional software development approach on mobile phones with the creation of services for BIT, we show in these case studies that our framework offers significant benefits.

1.6. Thesis Outline

This dissertation is structured as follows:

In Chapter 2, we will provide an overview of mobile device-mediated interaction with appliances. We will review existing work and ask which of the many proposed technical solutions are actually desirable from a user's perspective. We will then propose a classification of types of interaction tasks and form hypotheses regarding the suitability of mobile device-mediated interaction for the different task types. Finally, we will present the results of a user study that we conducted to confirm or reject these hypotheses.

In Chapter 3, we will begin by discussing scenarios that call for the standardized advertisement and lookup of information and services for tagged products. From these applications, we derive requirements and present the architecture of an infrastructure to meet them. We then outline a prototypical implementation of this infrastructure as well as example applications that exploit the advantages offered by the system. We conclude the chapter with a review of existing work in the field and investigate how the proposed infrastructure compares to these approaches.

In Chapter 4, we will start with an analysis of problems and challenges that arise when services for augmenting tagged objects are implemented and deployed on mobile phones — both from a developer as well as a user perspective. We then introduce the concept of a dedicated browser along with an underlying software framework to address these issues. After discussing its requirements, we present the framework's core concepts and show how they can be applied to developing services. We then present the architecture and implementation of a prototypical browser and nine example services. We contrast the development of these services with traditional approaches and show that our framework offers significant benefits. We also highlight the limitations of BIT and conclude with a review of related work.

2. Usability of Mobile Phones for Operating Appliances

2.1. Introduction

As we saw in the previous chapter, personal mobile devices, such as mobile phones and PDAs, have effectively become powerful general purpose computing platforms that are often equipped with different sensors. This allows their users to interact with an augmented, or "smart" environment. Scenarios involving personal mobile devices include attaching digital annotations to physical objects [19, 137, 124], sharing public displays [99, 9], and interacting with appliances of all sorts [10, 98].

Using mobile phones and PDAs to query and control smart environments and artifacts is attractive due to four main aspects of these devices:

- *Wireless communication:* Apart from the continuously expanding wide-area coverage, packet-switched communication services, such as GPRS, EDGE, or UMTS, have become commonplace and can provide fast, reliable, and economic device communication from almost anywhere in the world, both indoors and outdoors. Moreover, short-range communication protocols, such as Bluetooth, allow for local ad-hoc communication between similar devices.

- *Tag detection:* The recent addition of Near Field Communication (NFC) technology not only improves intra-phone communication (i.e., simplifying the often complicated Bluetooth setup process), but also allows mobile devices to detect and read out passive (NFC-compatible) RFID tags. Moreover, camera phones can use barcodes to allow even printed paper to "send" information to a mobile device.

- *Computational resources:* Mobile phones have become powerful computing devices, often featuring processors with hundreds of megahertz and considerable RAM and Flash memory. Given their energy demands for sustained wide-area communication provision, their powerful batteries can often easily support substantial calculations and short-range communications without significantly affecting the primary usage (e.g., telephone or organizer) of the device. Users are also accustomed to recharging their devices periodically, thus providing a virtually unlimited energy supply for locally installed applications.

- *Programmable screen and keyboard:* Many devices feature large color displays and programmable soft keys, 5-way navigation buttons, click wheels, or touchscreens, allowing system designers to quickly build a wide range of attractive and versatile interfaces. Built-in microphones and speakers can additionally support the use of speech commands and audio cues.

While it is obvious that personal mobile devices can provide the technical functionality needed to control smart environments, we believe that, from a user-centric perspective, their use cannot be justified in some of the commonly cited ubicomp scenarios. In particular, we challenge the notion of using personal mobile devices for *universal appliance interaction*, i.e., controlling technical equipment, such as vending machines, thermostats, or answering machines, through mobile phones. We contend that mobile phones offer benefits only in specific situations, but not in everyday use.

In this chapter, we discuss arguments supporting our claim and seek to answer the question under which circumstances it is appropriate to use a mobile phone for appliance interaction. We do so by presenting a user study that we conducted in order to assess the value of mobile phones as interaction devices for operating appliances.

This chapter is structured as follows: We begin with a review of related work that has suggested to use mobile devices for appliance control. In Section 2.3, we continue with a critical discussion of the usability of such scenarios and outline a pre-study that we conducted to gain experience with a specific scenario (Section 2.4). Section 2.5 presents our main study in detail, starting with the experimental design and participants, describing our apparatus and materials, and outlining our procedure. Section 2.6 contains the results of our study, both analytically and anecdotally. We close with a discussion and conclusions.

2.2. Related Work

The use of handheld devices for controlling the environment has already a long tradition, based on the (usually infrared-based) remote controls that provide access from afar to audio/video equipment such as TVs and stereos, but also lights, shades, garage doors, or even cars. Given the many remotely controllable appliances found in today's households, however, it is becoming less and less practical to have a separate remote control for each of them. Also, users increasingly need to carry out tasks that involve more than a single appliance, e.g., switching on the DVD player while also turning on the TV at the same time. Last but not least, many of today's remote controls are overloaded with functionality that users hardly ever need, resulting in large, unwieldy devices and hard-to-use interfaces.

A number of research projects (e.g., [60, 114, 110]), as well as commercial products (e.g., Philips Pronto[1] and Logitech Harmony[2]) have grown out of these needs. The simple approach taken by commercial products provides users with a conventional desktop application that allows them to design their own remote control by placing button widgets on a canvas mimicking a traditional remote control. The resulting user interface is then downloaded to a PDA-like, pen-based handheld, which uses traditional infrared transmission to communicate with the target device.

The approaches generally proposed in many research projects rely on the appliances providing a software interface that is rendered on the PDA's display for the user to interact with. The handheld is therefore *self-programming*. Using its wireless communication module (e.g., Bluetooth), the handheld transmits the user's commands to the appliance itself, or a proxy that in turn controls the appliance. Systems of this sort are presented by Hodes et al. in [60] and by Ponnekanti et al. with the ICrafter [114]. However, they are still based on laptops as interaction devices. The XWeb [110] infrastructure is similar, but the authors mention an implementation for pen-based devices without going into details. Common to these systems is the notion of abstract user interfaces to allow for device heterogeneity. By combining the abstract user interface description and a design template (a "stylesheet"), the handheld generates a platform-specific user interface that is pre-

[1]www.pronto.philips.com
[2]www.logitech.com/harmony

sented to the user. The concrete user interface is thus decoupled from the appliance.

Nichols et al. developed the idea further in the Pebbles project, where the authors use a PDA as a *personal universal controller* [105]. A similar, PDA-based system termed the *universal remote console* was proposed by Zimmermann et al. [147]. Contributions from both projects led to the standardization of the universal remote console framework within the INCITS V2 technical committee [77, 44]. The standard[3] aims to ensure interoperability between devices from different manufacturers.

Other projects have examined how interaction with household appliances can be extended to support natural language. An example is presented by Yates et al. [146], who propose a system that is based on speech recognition. Lieberman and Espinosa [83] build appliance interfaces around goals of users. They leverage knowledge mined from the OpenMind Commonsense knowledge base [136], a collection of 770,000 English sentences describing everyday life. Appliances sense the user's interaction (e.g., opening the microwave oven) and create a corresponsing English sentence (e.g., "I open the microwave"). Using the knowledge base, possible goals are inferred and offered to the user to choose. The selected goal is sent to a planner, which presents to the user all steps needed to satisfy the goal.

Another issue in this field is the selection of the active device. In environments with many remotely controllable appliances, it would be desirable if the system automatically detected the most likely target the user wants to interact with. Kaowthumrong et al. [67] use a Markov model to predict a likely appliance and present the corresponding user interface. This work is extended by Isbell et al. [64]. They try not to predict a single appliance, but to determine the task a user wants to accomplish. Using k-means clustering and Markov prediction, their system is able to adapt the presented user interface accordingly. The personalized user interface resulting from this approach is evaluated in a separate user study [111]. The authors compare it with interfaces that users have created manually in order to make them suit their needs best. Users could freely combine buttons from any device and also create special macro buttons that would trigger a sequence of actions, such as switching on both TV and DVD player at the same time. The study revealed that both approaches have weaknesses. While the interfaces

[3]www.incits.org/tc_home/v2.htm

manually designed by users often missed important buttons because people have incorrect models of their own behaviour, the machine learning approach requires a lengthy observation period to discover accurate models. A similar system is UNIFORM by Nichols et al. [106], which aims to generate user interfaces that look familiar to users by customizing their appearance based on heuristics as well as previously generated interfaces for the same user. Like the previous systems, SUPPLE by Gajos and Weld [48] aims to automatically generate user interfaces to control appliances. However, they treat the necessary user interface adaptation as an optimization problem to find the best arrangements of controls on the device available.

Finally, another line of research explores how a flexible configuration of multiple home appliances can be integrated. Examples of such projects are HUDDLE by Nichols et al. [107] and OSCAR by Newman et al. [101]. While these projects again mainly investigate the user interface aspect of such scenarios, others, such as InterPlay by Messer et al. [88] are more geared towards middleware aspects.

2.3. Usability Benefits

Virtually all of the papers cited in the previous section name a more or less similar set of appliances that are considered controllable through personal mobile devices. Among the often mentioned appliances are video recorders, DVD players, TVs, video projectors, stereos, answering machines, light switches, home security systems, ATMs, elevators, copy machines, cars, vending machines, heating control systems, microwaves, ovens, and washing machines.

Researchers have recently begun to look at the suitability of different possible interaction techniques for such scenarios, such as scanning a barcode, pointing with a laser beam, or touching an RFID tag [130, 4]. However, there is surprisingly little work addressing the question of which appliances are actually suitable for this new paradigm of interaction, and under which circumstances they are so. Koskela et al. [73] have studied the use of mobile phones, PCs, and media terminals in a household over six months. However, the handheld device could only be used to control lights and curtains in their setting. Rode et al. [121] conducted an ethnographic study to find out which household appliances users choose to program. Their research gives, however, no indication for which appliances a personal mobile device may be an appropriate interaction tool. Moreover, they do not consider spontaneous interaction with an appliance, but focus on the programming of actions for the future and the creation of macros to facilitate repeated tasks.

User studies investigating the performance of personal mobile devices for the spontaneous interaction with appliances were carried out by Nichols et al. [98, 104] as part of the evaluation of their personal universal controller [105]. They studied the use of a PocketPC to control a stereo and a telephone/digital answering machine. In particular, they compared the performance of 12 subjects when accessing the appliance either using the PDA interfaces, or the interface on the actual appliance. They used three metrics in their evaluation: time to accomplish a task, the number of errors made, and how much external help users asked for. For both the stereo and the phone, a list of tasks was created that had to be completed by each of the subjects. About two thirds of the tasks on both lists were chosen to be easy, i.e., requiring no more than one or two buttons pressed on the actual appliance. The more complex tasks involved five or more button presses. The time to accomplish a task was defined as the total time users needed to work through the complete task list. The authors found that, compared to the user interface on the PocketPC, interaction based on the physical appliance's interface took twice as long, entailed twice as many errors, and required five times more external help.

While these results seem very encouraging and supporting the vision of using mobile phones as universal interaction devices, they might strike one as somewhat counterintuitive: Why would a softkey-based interface of a generic PDA be more efficient for playing back voice messages of a *real* answering machine than the machine's own buttons? Why wouldn't the direct interaction with the physical machine help users with understanding and operating the device, by making use of the machine's *perceived affordances* [108]?

Obviously, using a PDA or mobile phone as an interaction device will be greatly beneficial when this represents the only way to access an otherwise invisible device, e.g., for information services such as voicemail systems or online media libraries [51]. Similarly, a universal interaction device might be the only means for users to discover the invisible information services available in a smart room or attached to a smart object. And obviously, as the success of the TV remote control has shown, handhelds should be well suited to control appliances where interaction at a distance is desirable, such as a heating control system.

However, it is less clear whether personal mobile devices are beneficial for interacting with physical appliances that require the user's presence to operate, such as ATMs, elevators, or microwave ovens.

We believe that, for many of the appliances mentioned in typical ubicomp papers, including the ones listed before, it is not advisable to use handhelds as interaction devices in order to replace existing physical user interfaces. In most everyday situations, direct manipulation of the appliance is easier, faster, and more convenient than handheld-mediated interaction.

In *special situations*, however, this approach can be of great value. By special situations we mean processes that are performed irregularly and rarely. In these cases, users often face one of the following problems: First, they lack the practice needed to remember the individual steps that have to be performed in order to achieve a specific goal. Second, functions that are not accessed on a regular basis are often not present on an appliance's main user interface, both to simplify the interface for common tasks, as well as to save costs. Interaction is often difficult, as keys change their meaning in special contexts, or special functions are accessible only by a hard-to-remember combination of keys. As the following examples illustrate, we face this problem with several devices in our daily lives:

- Many ovens can be programmed to start cooking a meal at a user-given time. However, as this function is rarely accessed, it is hard for users to remember the programming procedure.
- Many washing machines offer several programs for a given temperature. It is often difficult for users to remember the exact semantics of buttons labeled "40°", "40°S", and "40°E", for example.
- Many appliances, such as laser printers or VCRs, show an error code on a small display when a problem (e.g., a faulty network interface) occurs. Without consulting the appliance's long-since lost manual, this error code (e.g., "F602") is incomprehensible and of little value for a user.

In contexts that users encounter infrequently, just like the ones outlined above, a personal mobile device can facilitate the user's interaction with the appliance in two ways. The handheld can either *provide information* or *provide a user interface.*

Information provision When an exceptional situation occurs, the appliance can support the user by providing detailed information on his or her personal mobile device. For example, by opening the relevant section in the appliance's manual on the user's mobile device, a laser printer can instruct him or her to check the network cabling instead of just showing an error code on its integrated display.

User interface provision Functions that are rarely needed and are thus not easily accessible through the actual appliance's user interface are offloaded onto the mobile device, *without* completely replacing the traditional user interace. In this way, a GUI based on familiar widgets can be built to, for example, program an oven's timer, while keeping the haptic user interface for the everyday task of switching the oven on and off.

2.4. Pre-Study

With this in mind, we set out to conduct a user study exploring the benefits and limits of using a mobile phone to operate physical appliances, i.e., devices that would typically not benefit from being remotely controllable. Our aim was to identify in particular the conditions under which devices like coffee makers, printers, or microwave ovens would benefit from being operated not directly, but through a mobile phone, or, conversely, when it would be a hindrance, rather than an advantage, to have such a separate interaction device.

Our user study was conducted in two parts. The *main study* was preceded by an *exploratory phase* that was concluded by a small pilot study to test and improve the planned study design. We began this exploratory phase by developing a series of scenarios for the use of mobile phones to operate everyday appliances. We then broke down these scenarios into smaller use cases and also involved a student of the Zurich University of the Arts (ZHdK) who provided valuable design prototypes of possible user interfaces (see Figure 2.1 for an example). From the many possible appliances, we finally selected a coffee maker for our pilot study, since we felt it was an ideal example to communicate the benefits of mobile phone-mediated interaction to our study subjects.

We then implemented several versions of devices that could be used to control the coffee maker:

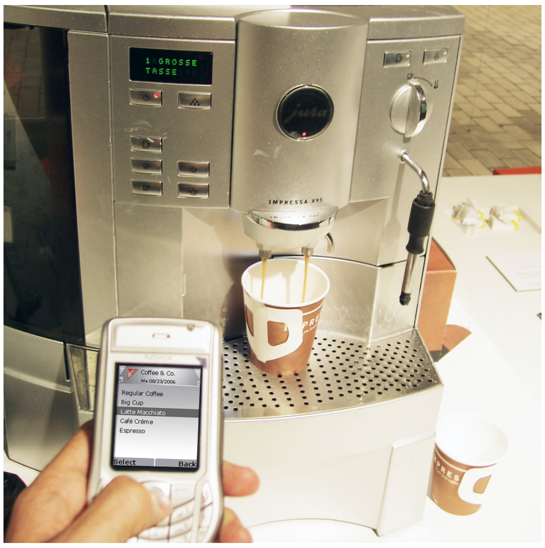
- A version based on Adobe's Flash Lite technology for the Nokia 6630 mobile phone that explored the design possibilities offered by the platform and focused on an appealing visual appearance (see Figure 2.1).
- A version based on Java ME for the Nokia 6630 mobile phone that included Bluetooth connectivity to a simulated coffee maker.
- A second version based on Java ME for the Nokia 3220 NFC mobile phone that made use of the device's integrated Near Field Communication (NFC) module (see Figure 2.2). This allowed us to read RFID tags that we attached to the coffee maker to implement the *touch me paradigm* [113], through which users can interact with appliances in their proximity in a very intuitive way. Also, NFC can be used to transmit status information directly from the appliance to the mobile phone. However, we did not employ this feature, but relied on NFC merely for the identification of the appliance.

The pilot study involved 7 participants, aged between 24 and 33 years (5 males, 2 females) and required the subjects to interact with a Jura Impressa S70 coffee maker through our software running on a Nokia 3220 NFC phone. Subjects were asked to complete a series of five tasks, such as brewing a "latte macchiato" or replacing the water filter, and to provide feedback. The pilot study revealed some misunderstandings on behalf of our test users as well as measuring and interpretation problems. We used these learnings to prepare an improved design for our main study, which will be presented in the next few sections.

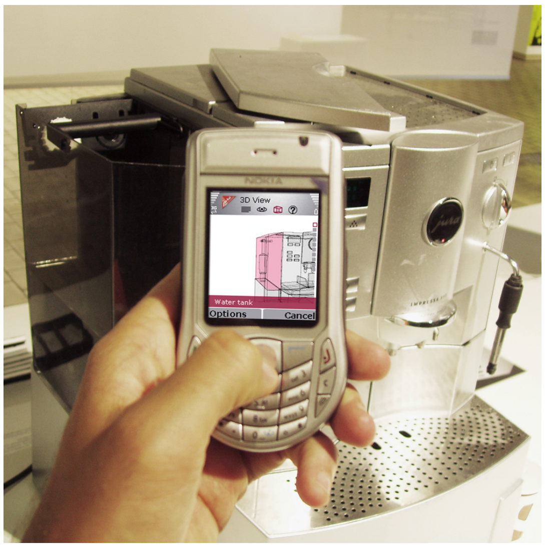
2.5. Main User Study

Based on the preliminary experiences and observations gained in the exploratory phase, we proceeded with the main study. Our main study tried to assess the benefits and limits of handheld devices in appliance operations by asking study participants to use not just one, but several different appliances in a variety of situations, both traditionally through the appliances' physical interface, and with a specifically developed universal interaction device. We then obtained quantitative data by measuring the time it took a test subject to complete a specific task, as well as qualitative data by observing and recording the users' actions,

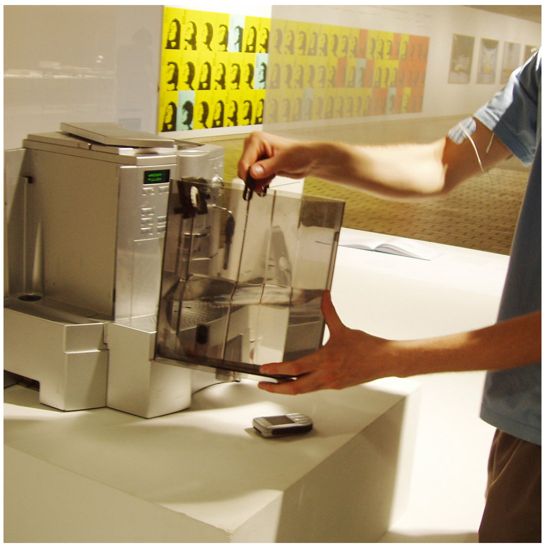
(a) User interacting with coffee maker.



(b) Brewing a "latte macchiato".



(c) Instructions for filter change.



(d) Changing the filter.

Figure 2.1.: Flash Lite-based design prototype developed in exploratory phase [49, 96].

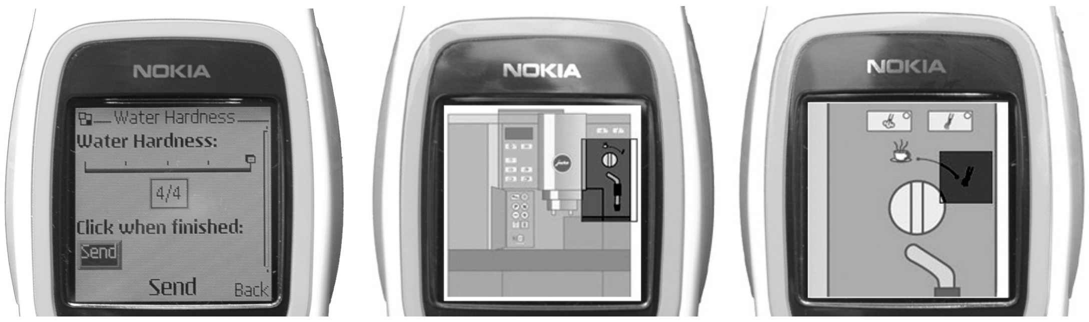


Figure 2.2.: NFC-based prototype developed in exploratory phase.

thoughts (through think-aloud techniques [82]) and opinions (through a post-test questionnaire). This section presents the hypotheses, tasks, and procedure of our study, including a description of our prototypical universal interaction device, the AID (short for “Appliance Interaction Device”).

2.5.1. Hypotheses

We began our main study with a set of three hypotheses, which together would either support or weaken our intuitive notion that the use of universal interaction devices has limits. Specifically, we hypothesized as follows:

- For controlling an appliance in exceptional situations, interaction based on a mobile phone would be faster than interaction based on the traditional user interface.
- Looking up context-dependent information on the handling of an appliance would be faster using a mobile phone than using traditional means (e.g., printed user manuals).
- To carry out everyday tasks, the use of an appliance’s traditional user interface would be faster than mobile phone-based interaction.

2.5.2. Appliances and tasks

We used four typical appliances (see Table 2.1) for which we found a number of use cases where mobile phone-based interaction might be beneficial: a dishwasher (V-ZUG Adora S 55), a coffee maker (Jura Impressa S70), a printer (HP LaserJet 4050tn) and a radio (Sangean ATS-505). By not relying on a single model or appliance type, we aimed to minimize a possible bias arising from poor user manuals and haptic user interfaces. When selecting the concrete models of appliances for our study, we made sure to only consider devices that seem reasonably well designed from a usability point of view. This is emphasized by the fact that the majority of devices used in the trial are products of well recognized manufacturers with a proven track record in appliance design.

For each appliance, we defined a number of tasks for participants to work through – once using the appliance’s native controls, once using our AID device. We grouped these tasks (18 in total, listed in Table 2.2) into the following four categories:

Coffee maker *Jura* *Impressa S70*		This coffee maker model is common in households as well as small offices. The grinding and brewing processes are fully automated. Status information is shown in a small display of 2 × 8 characters. Apart from the controls for daily use, this coffee maker has eight buttons for configuration that are covered by a bezel in normal operation.
Laser printer *HP* *LaserJet 4050tn*		This black-and-white laser printer can be found in many small and medium offices. It can be controlled through a four-button panel ("Menu", "Option", "Value", "Select") and a display that is capable of showing two lines of 16 characters each.
Dishwasher *V-ZUG* *Adora S 55*		The V-ZUG Adora S 55 is a widely used dishwasher with a 4-digit seven-segment display. Users can navigate the settings menu by pressing the "start program" and "end program" buttons.
Radio *Sangean* *ATS-505*		This portable radio is able to receive radio programs in several frequency bands. The current frequency is shown numerically on the LC display. The device's functions are accessed through a number of buttons.

Table 2.1.: Appliances used in the user study.

- *Control tasks* involve the adjustment of a special device setting (e.g., setting the water hardness for the coffee maker) or the invocation of an unusual operation (e.g., create a printer cleaning page). Theses tasks reflect the use of a mobile phone for user interface provision.
- *Problem solving tasks* confront users with an abnormal situation (e.g., a malfunctioning dishwasher displaying an error code) that must be dealt with. These tasks correspond to the use of a mobile phone for information provision.
- *Everyday tasks* are control tasks that are most typical for an appliance (e.g., brewing a coffee) and are performed very often.
- *Repeated control tasks* are control tasks that a user has performed very recently and is still very familiar with.

When we identified the tasks to be included in our study, we made sure to consider only those suitable for comparison. For example, the coffee maker's manual contained instructions on how to brew a latte macchiato, which required the user to adjust various settings in no less than 10 steps. Obviously, brewing a latte macchiato could simply be offered on the mobile device as a single menu item. As this would have drastically favored the AID device, we took care to select only tasks that would require a comparable degree of interaction when executed directly on the appliance and on the mobile interaction device. Similarly, we omitted tasks that were so poorly supported by the appliance manufacturer that they proved very difficult and lengthy when tested in

Table 2.2.: *Appliance tasks in the user study.* Participants were asked to complete 18 tasks distributed among the four available appliances. Not all appliances had suitable tasks in all four categories.

	Control	**Problem solving**	**Everyday**	**Repeated control**
Dishwasher	adjust water hardness, activate child safety lock	fix error "F2", white film on dishes	start program	—
Coffee maker	adjust water hardness, set switch-on time	—	brew coffee	adj. water hardness
Printer	change paper type, print cleaning page	fix faded print	cancel print job	change paper type
Radio	set clock, store preset station	—	change channel	set clock

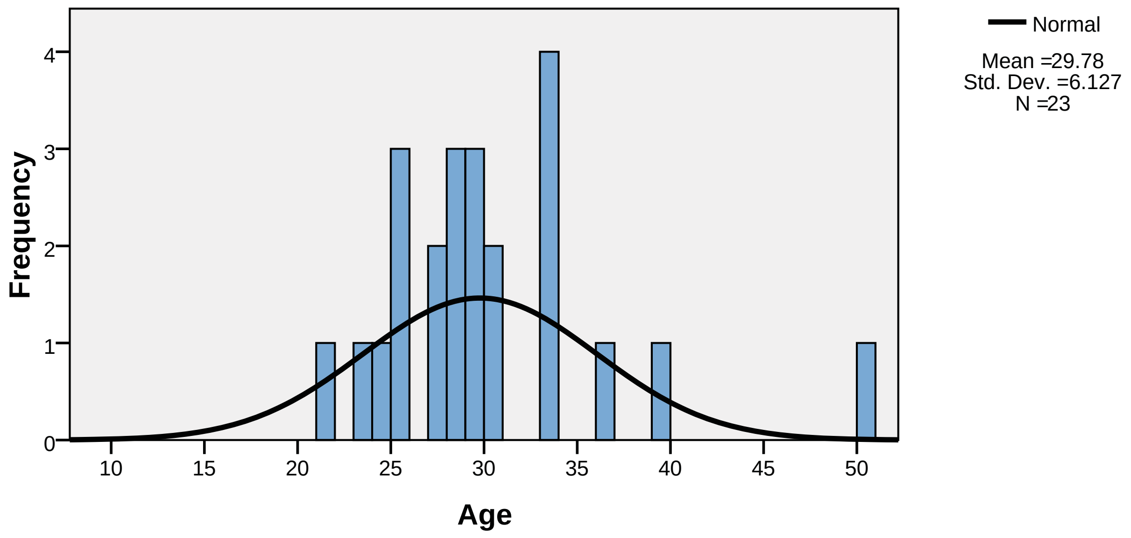


Figure 2.3.: Distribution of participants' age.

our pilot study, e.g., changing the coffee maker's water filter. As these tasks could be improved easily (e.g., through better documentation), we did not consider them for our main study.

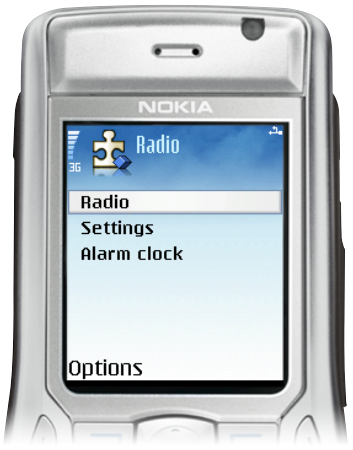
2.5.3. Participants

We tested a total of 23 participants,[4] 10 (43%) of which were male and 13 (57%) of which were female. Most of them were undergraduate or graduate students recruited through mailing lists from different universities. There were 10 participants with a background in sciences or engineering, 10 participants from the humanities or social sciences, and 3 participants with no academic background. All of them spoke Swiss German as their native language and owned a mobile phone. Except for two subjects, all participants used their mobile phone on a daily basis. The average age of participants was 29.8 years, ranging from 21 to 50 (SD=6.1) years (see Figure 2.3). None of them had any relevant previous experience with the appliances used in our experiment. Participants were compensated for their time with a USB memory drive or a cinema voucher, according to their preferences.

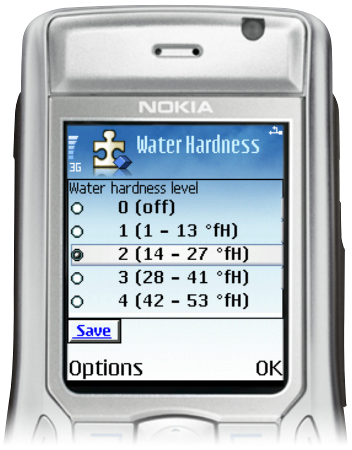
2.5.4. Apparatus

In order to evaluate the usefulness of a universal interaction device, we had to provide our test subjects with a corresponding mobile unit that would let them properly experience the use of such a control unit.

[4]None of these subjects had participated in our earlier pre-study.

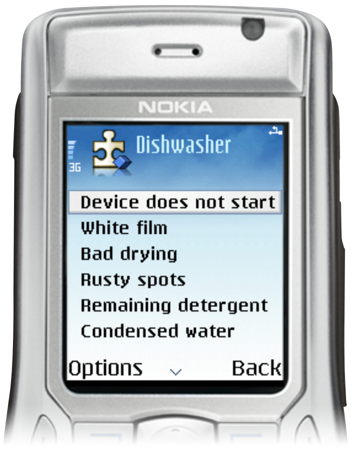


(a) Radio main menu.

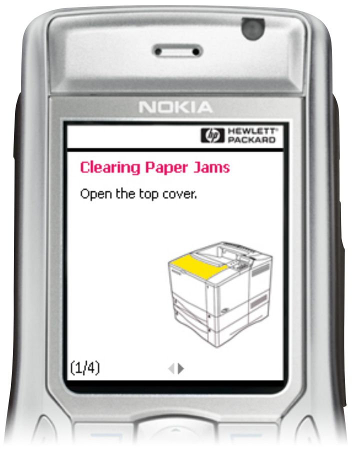


Water hardness level
- ○ 0 (off)
- ○ 1 (1 - 13 °fH)
- ● 2 (14 - 27 °fH)
- ○ 3 (28 - 41 °fH)
- ○ 4 (42 - 53 °fH)

(b) Adj. water hardness.



(c) Dishwasher user manual.



Clearing Paper Jams

Open the top cover.

(d) Printer troubleshooter.

Figure 2.4.: *AID prototype implementation.* The screenshots show examples of the AID user interface for each of the four appliances used in the study. Appliance functions could either be selected by traversing the main menu (a) or by having the AID automatically (i.e., simulated, see Section 2.5.4) detect the appliance's state (d).

Our AID prototype system supports mobile phone-based interaction for all of the tasks outlined above. Our system is implemented as a Java MIDlet that we deployed on a Nokia 6630 mobile phone. This mobile phone features a color display with a resolution of 176 × 208 pixels. Unlike the devices used in other evaluations (see Section 2.2 above), the Nokia 6630 does not offer pen-based input capabilities, but features only a simple keypad.

While the mobile phone used in our pilot study was an NFC-enabled Nokia 3220 that actually performed a true wireless identification of each appliance, we did not stick with this device for the main study. Our earlier experiences had shown that the computational resources of the Nokia 3220 were too limited in comparison with the Nokia 6630. Also, the 6630 allowed us to take advantage of its larger screen estate.

Instead of actually coupling the AID with our four appliances, we simulated both appliance identification as well as the transmission of appliance status by having the user press the phone's main navigation button. As users performed the tasks in separate steps, the experimenter had time to use an undocumented button combination on our AID to quickly select the proper context-dependant reactions for the upcoming task, thus giving users the illusion of having our AID actually detect and read out the appliance. Obviously, this setup made it impossible to really *control* the appliance in any way through the AID – a shortcoming that we pointed out to participants, indicating that we were only interested in seeing the right buttons being pressed, not an actual appliance being controlled.

Figure 2.4 shows four example screenshots of the AID during the study, one for each of the four appliances. Invoking our AID device on an idle appliance brings up the appliance's main menu, as shown in Figure 2.4(a). For each appliance, this main menu would offer all tasks that are available through the appliance's physical interface (in a hierarchical menu). The user interface for a typical task is shown in Figure 2.4(b). We also included a "troubleshooting" menu entry for each task, which would contain the contents found in the corresponding section of the appliance's user manual. Figure 2.4(c) shows such a list of common problems that might occur with the dishwasher. Finally, we provided several step-by-step assistants that help users with physical appliance manipulations. The assistant supporting the task of clearing a paper jam at the laser printer is illustrated in Figure 2.4(d). For each step, the system highlights the part of the printer the user must operate next. When the user pushes the right arrow on the phone's keypad, the assistant advances to the next step. Assistants and troubleshooting tips can either be accessed manually through the menu, or they are displayed automatically when the appliance is in a state where such help is needed.[5]

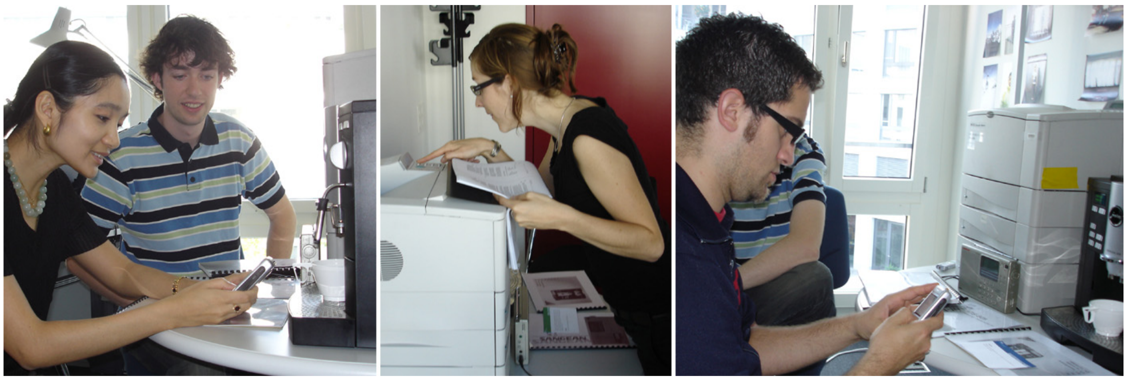


Figure 2.5.: *Participants interacting with appliances.* The images above show three of our participants carrying out tasks: using the AID device to operate the coffee maker, troubleshooting the printer using the printed manual, and using the AID device to operate the radio.

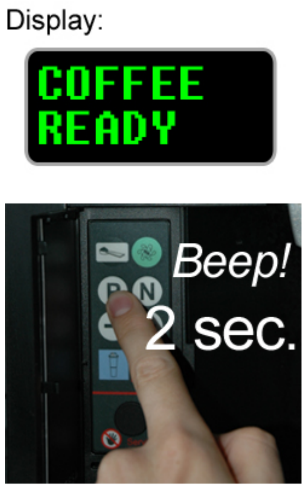
2.5.5. Procedure

The experiment began for each participant with a brief introduction to the concept of the AID, our user study, and its goals. Participants were then asked to fill out a profile questionnaire that allowed us to gather information on their background (age, education, previous experience with devices used in the experiment, etc.). We explained the basic concepts of the AID (i.e., the user interface provision and the information provision) and demonstrated it interactively using example tasks that did not reoccur in the course of the study. We also told participants explicitly not to think of the AID as a remote control, and that they would only be able to use it on an appliance after they had touched it, followed by pressing the phone's main navigation button. We finally handed them the AID device and guided them through a number of simple preparation tasks in order to familiarize them with the phone's controls and user interface.

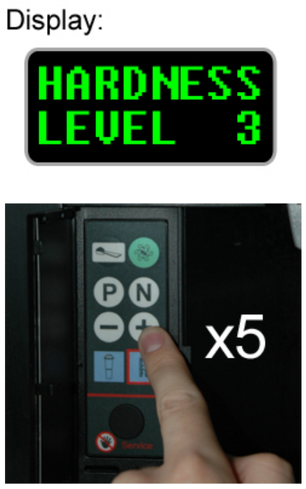
The beginning of the study was conducted in a semi-public lounge in our university, as it offered a dishwasher that we could use. There were only few distractions here, and for the rest of the study we moved to a quiet office where we had set up the coffee maker, the laser printer, and the radio. The introduction and initial explanations described above were also conducted in this office. All devices where ready to use and equipped with the corresponding user manual in German language.

For each of the four appliances, we handed the test subjects small

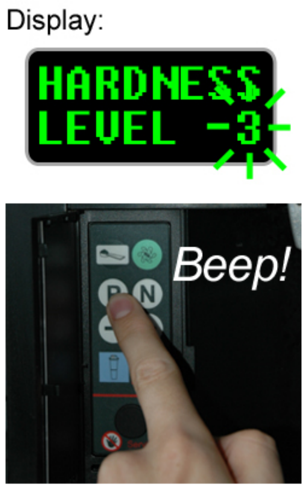
[5]As pointed out previously, this appliance state detection would be set up secretly by the experimenter prior to giving out a particular task to the user (as no communication between the appliance and our AID device takes place).



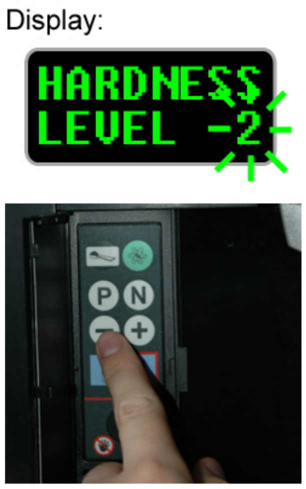
(a) User enters the menu by pressing and holding the “P” button for 2 seconds.



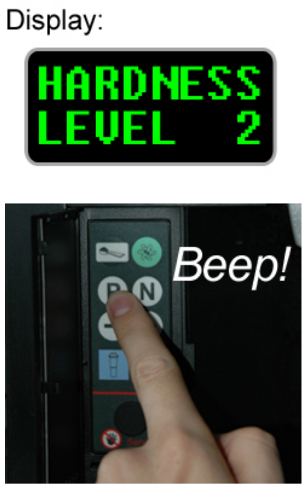
(b) User selects the desired menu item by pressing the “+” button repeatedly.



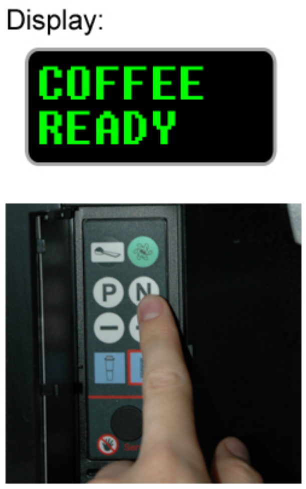
(c) User opens the menu item to adjust the value.



(d) User presses the “-” button to set the water hardness to level 2.



(e) User presses the “P” button to save changes.



(f) User presses the “N” button to leave the menu.

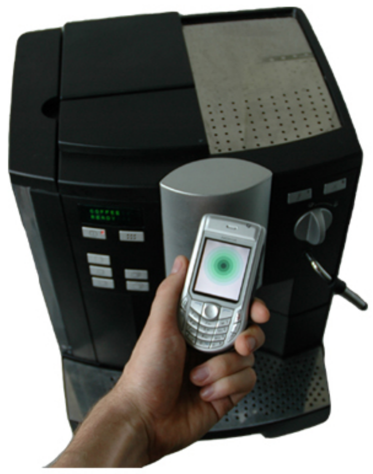
Figure 2.6.: *Example of traditional task solving.* At one point, study participants were asked to change the level of water hardness in the coffee maker. The above pictured steps usually required an extensive study of the printed manual.

cards with their assignment printed on (see Appendix A). They were asked to work through each task twice. One time, users should use the traditional method to solve the task, i.e., they should interact directly with the device using the physical interface. The other time, they should use the AID device. Users were explicitly told that they could, but would not have to use the user manual when completing a task in the traditional way.

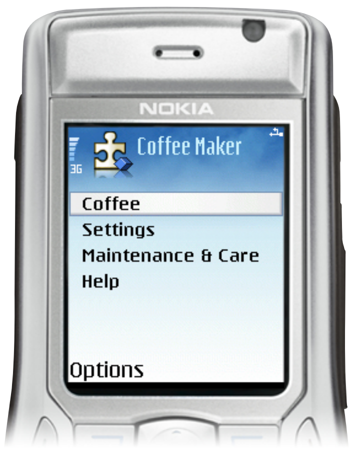
In order to minimize potential learning effects, we used counterbalancing, i.e., participants were divided into two subgroups that worked through the cards in different orders. Group A was asked to complete every task first with the traditional method, and then again with help of our AID device. Group B was asked to begin with the AID device and then use the traditional interface afterwards. To get comparable results, the order of the tasks was the same for all participants. The actual task order can be found in Table 2.2: for each device, the control, problem solving, and everyday tasks were performed (in this order). Finally, the column "repeated control" was tested from top to bottom. Learning effects were compensated for by letting the users perform the first task of each appliance again at the end of the study. For practical reasons, we did not move back to the lounge area again to test the dishwasher a second time.

We measured the time needed for each task and then asked users whether it was easy or difficult for them to solve the task, as well as which method they liked better. For tasks that required users to find a solution to a problem and telling us about it, time stopped with their first utterance. Figure 2.5 shows some of our participants carrying out tasks, Figures 2.6 and 2.7 show an example of an entire task (adjusting the water hardness in the coffee maker) being carried out using the traditional method and using the AID device, respectively.

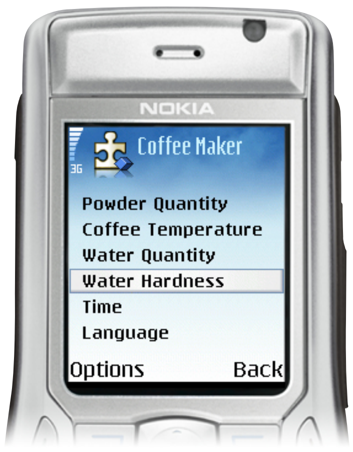
The final part of our study asked our participants to complete seven different tasks in a row (see Figure 2.9 on page 35 for a list), which they had to perform as fast as possible. They were, however, free to choose any method to operate the appliances, i.e., they could pick up the AID, consult the printed manual, or directly operate the physical appliance interface to accomplish the assignment. We then recorded their individual choices for each of the seven tasks. In order to get participants to truthfully choose the methods they thought would be most effective, we offered a portable MP3 player to the user with the fastest time.



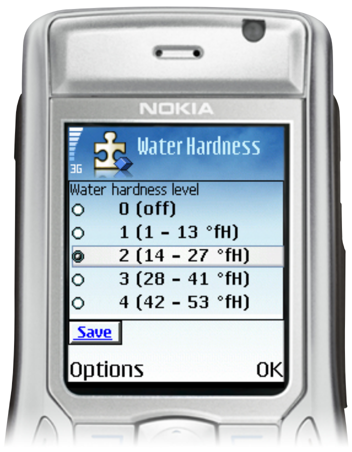
(a) User touches the appliance with mobile phone.



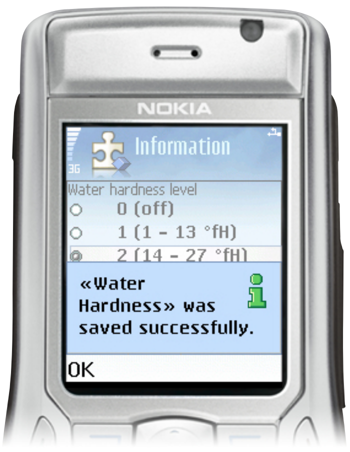
(b) Menu shows up automatically. User selects “Settings”.



(c) User selects “Water Hardness”.



(d) User selects the desired value and saves changes with “OK”.



«Water Hardness» was saved successfully.

(e) Setting is changed on the appliance.

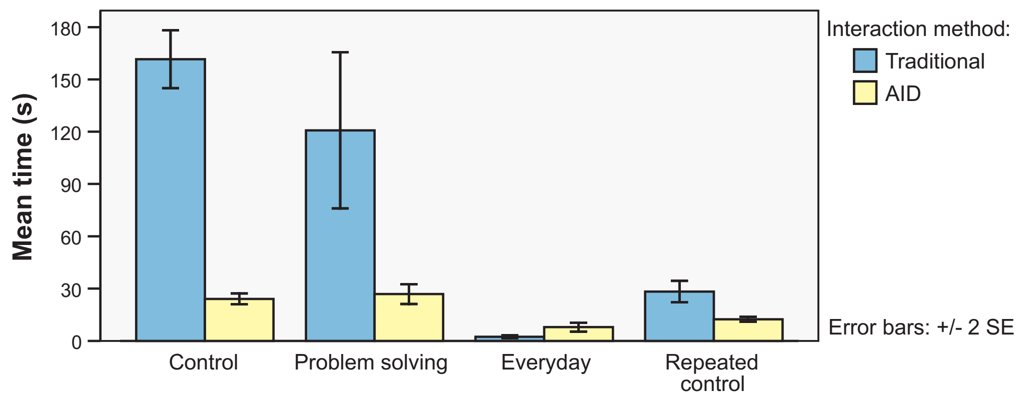
Figure 2.7.: *Example of task solving using the* AID. The water hardness can be changed more easily using the hierarchical menu of the AID, typically without consulting the user manual.

The study ended for each participant with a final questionnaire that collected her or his opinion and suggestions on the AID device.

A single session typically lasted about 80–120 minutes. The protocol followed in all sessions is summed up by the following list:

1. Welcome and introduction to the idea of an AID device as well as to the user study and its goals.
2. Consent form and profile questionnaire.
3. Demonstration of how AID can be used to control appliances.
4. Preparation tasks carried out by participant for getting to know mobile phone and user interface.
5. For each appliance, i.e., dishwasher, coffee maker, laser printer, radio (in this order):
   a) Hand the participant a small card describing a new task (see Appendix A).
   b) Tell the participant to carry out the task using the traditional user interface (for participants in group A) or the AID (for participants in group B).
   c) Measure time to complete task.
   d) Ask participant to rate the difficulty on a five-point Likert scale.
   e) Tell the participant to carry out the same task again, using the other interaction method (AID for group A, traditional user interface for group B).
   f) Measure time to complete task.
   g) Ask participant to rate the difficulty on a five-point Likert scale.
   h) Ask participant which method she or he liked better.
   i) Next task (order according to Table 2.2).
6. Tell the participant to again carry out the first task for each appliance (repeated control task).[6]
7. MP3 player contest with free choice of interaction method.

[6]For practical reasons, this step was omitted for the dishwasher, as this appliance was located in a different area of the building in which we conducted our study.



Task type	Traditional		AID	
	μ	σ	μ	σ
Control	162	112	24	21
Problem solving	121	131	27	16
Everyday	2	4	8	12
Repeated control	28	24	12	6

Figure 2.8.: *Mean time (in seconds) and standard deviation for task completion.* While the use of the AID device significantly cut down the execution time for exceptional control and problem solving tasks, it was two to four times slower for everyday tasks.

8. Post-test questionnaire.

9. Handing over of present to compensate for time.

2.6. Results

We collected both qualitative and quantitative results. Qualitative results used both the explicit answers from a post-test questionnaire administered after all tasks were completed (reported in Section 2.6.2), as well as notes and recordings taken during the tasks that captured the participants' thoughts through a think-aloud technique, which they were instructed to employ (see Section 2.6.3). The quantitative results simply measured the time between the start and completion of a task, though for tasks involving an answer rather than asking the user to operate an appliance (i.e., the problem solving tasks, see Table 2.2 on page 23), the time until the user started to reply was measured. This data is reported in the following section.

2.6.1. Quantitative results

Figure 2.8 shows the average task completion times we measured in each condition. As predicted, the mean time of task completion decreased for control tasks and problem solving tasks, whereas it increased for everyday tasks.

We further examined the collected data using analysis of variance (ANOVA). We ran a two-way ANOVA with factors interaction method and task type, which confirmed a significant main effect of interaction method ($p < .001$, $F_{1,736} = 100.23$). Also, a significant main effect of task type ($p < .001$, $F_{3,736} = 661.18$) and a significant interaction effect between task type and interaction method ($p < .001$, $F_{3,736} = 218.68$) was found. Focusing on this interaction effect, we continued by analyzing the effects of interaction method for each of the four task types. For each task type, an ANOVA showed that there was a significant difference between interaction methods (control tasks: $p < .001, F_{1,366} = 600.1$; problem solving tasks: $p < .001, F_{1,64} = 34.823$; everyday tasks: $p < .001, F_{1,182} = 198.302$; repeated control tasks: $p < .001, F_{1,124} = 44.851$). We therefore find our hypotheses confirmed that mobile phone-based interaction significantly reduces completion times for control and problem solving tasks, while significantly slowing down everyday tasks. Interestingly, controlling an appliance with the mobile phone was significantly faster even after participants had familiarized themselves with the task and could be considered experienced users of the respective appliance.

We also studied the effects of other factors, namely age, gender, and experience with advanced phone features. We did not find an interaction effect with interaction method for any of these factors. We therefore conclude that, for the 23 participants in our study, the use of the AID was beneficial irrespective of their gender, age, or previous mobile phone experience.

An individual analysis of our 18 tasks showed a consistent pattern. For each of the everyday tasks, traditional interaction proved faster, while users were faster using the AID in all other cases. However, there was a single exception from this picture, namely the repeated control tasks performed on the printer. While interaction with the AID took slightly less time ($M_{AID} = 16.9s, M_{trad} = 21.6s$), this difference fell short of significance at the .05 level ($p = .177, F_{1,40} = 1.885$). We presume that this is due to the well designed user interface of the printer used in our study. Several users made the informal comment that they liked the layout of the printer's control panel and that it was relatively easy to navigate in its menu because buttons were labeled in a helpful and familiar way.

Apart from the repeated control tasks, we found the results of the problem solving tasks particularly interesting. While in one task the appliance supplied context (i.e., an error code) that the AID used to automatically show the relevant instructions, it did not do so in the two other repeated control tasks. We merely gave participants some unspecific information about a problem that they had to find manually using either the printed user guide or the AID. Even though we made sure the AID covered all the topics we had found in the printed documentation, participants were significantly faster when they used the AID (printer: $M_{trad} = 253s, M_{AID} = 25s$; dishwasher: $M_{trad} = 91s, M_{AID} = 29s$). This is surprising especially for the dishwasher, as its user manual is very compact and, as we find, well made. However, the mobile phone's menu hierarchies that allow for the structuring of content seem to prove beneficial for this task.

2.6.2. Qualitative results

Figure 2.9 summarizes the results of the final contest, in which participants had to solve a list of tasks in the shortest possible time, but for which they could freely choose the interaction method. Overall, the AID was used in 69% of the control and problem solving tasks, even when participants had previously used the traditional user interface in similar situations just as effectively as the AID. Most of the participants stated that already slightly different tasks would make them insecure, fearing that they would not know where to find it in the appliance's menu structure. They would therefore opt for the AID, as it "*gives me a better overview than the printer's two-line display and allows me to complete the task faster*", as one participant explained. On the other hand, participants hardly used the AID to store or select a station in the radio's preset memory, stating that they didn't even think of using the AID, as it was "*just more natural*" to interact with the radio directly.

In the post-test questionnaire, we asked users to rate a number of statements on a five-point Likert scale. The statements and participant responses are shown in Figure 2.10. We also asked users to rate the value of the AID in some given situations. The results of this question

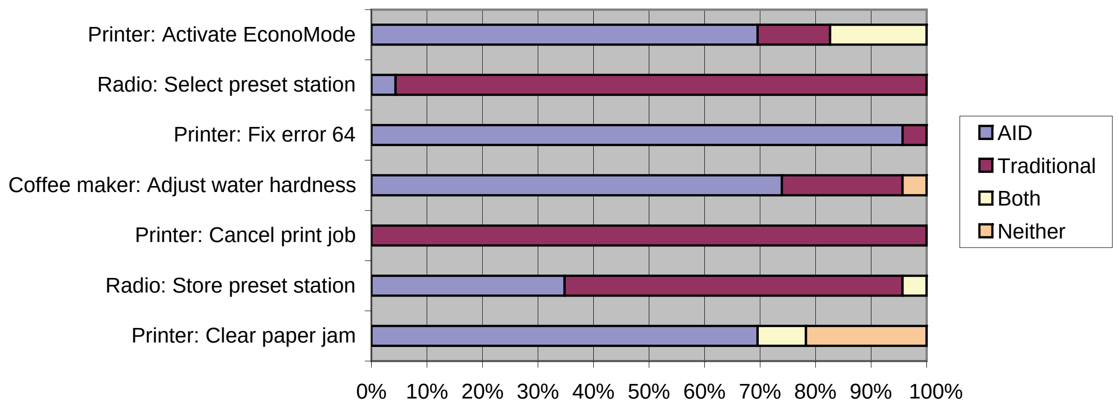


Figure 2.9.: *Interaction method usage during contest.* For the final contest, participants were free to use any method that they felt most comfortable with for each taks. Out of the seven tasks, four were previously completed tasks, while three ("Activate EconoMode", "Fix error 64", "Clear paper jam") were new tasks.

are depicted in Figure 2.11.

Finally, every participant was asked to answer three concluding questions. At the outset, we asked them: "*Do you use appliances for which you would like to have the AID? If so, please name these appliances.*" 22 participants (96%) replied positively and listed both devices from our study, as well as microwave ovens, TVs, DVD players, car radios, heating systems, and "*appliances that you change often, such as rental cars*". Only 1 participant (4%) would not want our AID. We also asked: "*If you owned an AID for an appliance, would it be ok if you could access rarely used functions* only *through the AID? Why / why not?*" 18 (78%) of our users agreed, 4 (17%) disagreed, and 1 participant was not sure. Some participants expressed concern that they would no longer be able to interact with their appliances in case their mobile phone or PDA was unavailable.

We ended our questionnaire with the following question: "*Could you imagine accessing* all *functions offered by an appliance only through the AID? Why / why not?*" While some participants could imagine giving up the traditional interface (4 users, i.e., 17%), most replied that a software-only user interface was not an option for them (17 participants, i.e., 74%). 2 participants were unsure. Most users answered that they would rather not use the AID for simple tasks. However, two respondents stated that they might be willing to accept an appliance without a traditional user interface if its price were lowered in turn. Two participants added that they could think of appliances that would benefit from a full replacement of the user interface, as this would make

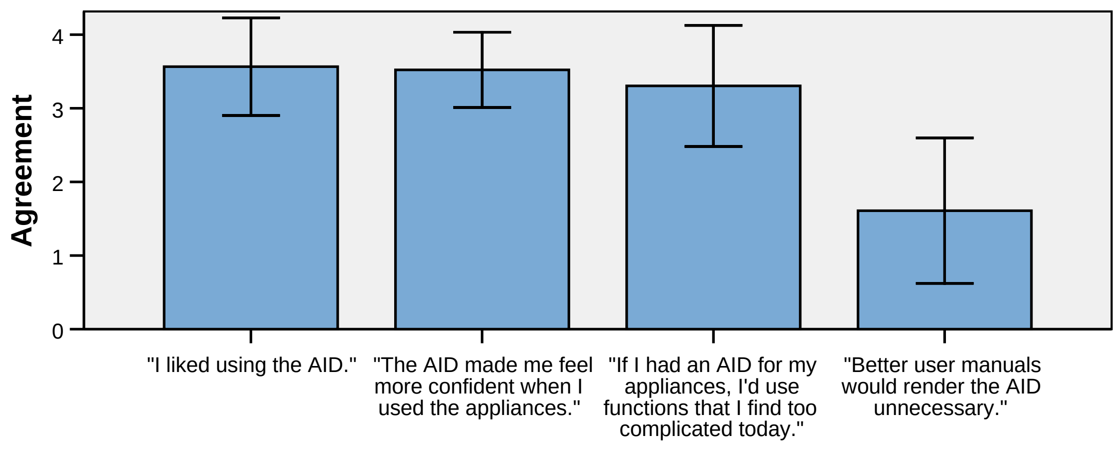


Figure 2.10.: *Subjective Likert scale responses.* Participants rated the usefulness of the AID device after they finished all tasks (0 = strongly disagree, 4 = strongly agree) — Error bars: +/- 1 SD.

ugly control panels redundant and improve visual appearance.

2.6.3. User feedback

In the course of the study, there were a number of issues that were brought up by the participants. In this section, we review these informal comments.

The biggest concern that users expressed was that of increasing dependence on technology, and in particular the mobile phone, through the use of an AID device. Most often, participants wondered how they would use their appliances if their phone was misplaced, lost, stolen, malfunctioning, without network coverage, or had simply run out of battery. Two participants also mentioned that they did not want to always carry their phone with them, just to be ready to use their appliances. Other concerns were more diffuse: "*I don't fully trust the mobile phone and would like to have the buttons on the device. Just in case...*" Someone else perceived the AID as yet another burden in everyday life: "*We've got so many devices around us already, so I don't think we need another one just for rarely used functions.*"

What most participants pointed out as the AID's biggest advantage over traditional interfaces was its menu structure. It was described as "*easy to use*", "*well structured*", "*clearly laid out*", and "*intuitive*". Many participants felt that this was mainly due to the larger display size, especially when compared to a small appliance LCD. Several participants explained that they would hardly get lost in the menu, thus enabling them to easily find even previously unknown functions. One

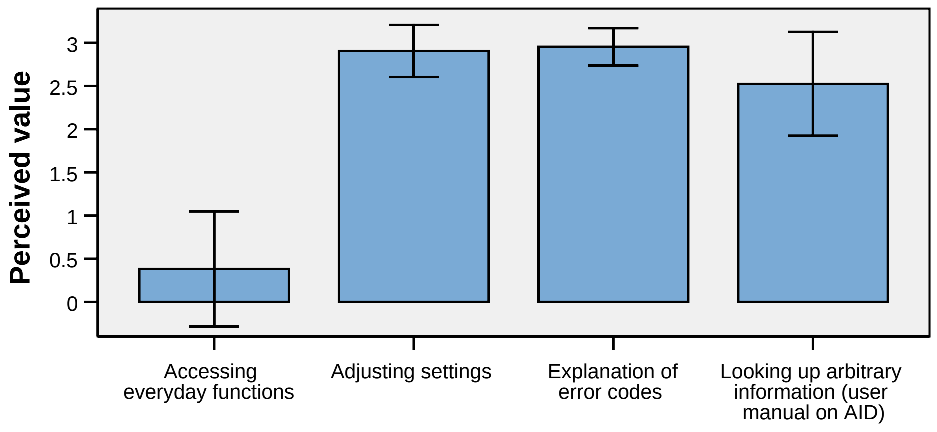


Figure 2.11.: *Concluding questions.* Participants were asked to rate the perceived value of the AID device in various situations (0 = no value, 3 = great value) — Error bars: +/- 1 SD.

participant said: "*If I had an AID, I could forget about how I did something and still be able to do it a few months later without any problems.*" Another participant explained how the AID's menu system allowed her to interactively explore the full range of functions: "*I always like to gain an overview of all available functions of my appliances. The AID would be ideal for that.*"

While they liked our prototype implementation of an AID, some participants expressed doubt if an actual appliance manufacturer would be able to come up with an equally user-friendly piece of software for their own products. One participant suggested that "*every manufacturer should include a menu item called 'troubleshooting', just like all Windows applications have a 'Help' menu.*" Many users especially liked the immediate reaction of the AID to special situations, e.g., by displaying helpful instructions instead of just a status code. As one participant put it: "*It is also extremely convenient that I'm immediately told what the cause of an error is.*" However, there were also cases where context-sensitive behavior confused people more than it helped, as they expected the normal menu to show up.

Most participants expressed their frustration with user manuals. As one user said: "*[Most handbooks contain] too much information that I don't actually want to read.*" They therefore found it very helpful to have the essential problem- or task-oriented information available on the AID and were not bothered by its relatively small display size. We asked some participants if a well made quick-reference guide could make this content obsolete on the AID. Most of them agreed that this was not the case: "*I don't want to go find a manual that I anyway* *may have thrown away.*" Two participants also said that the AID could be improved by adding search capabilities because "*then I can search for keywords with CTRL+F just like when I read a manual in PDF format.*"

Overall, while having some concerns on the ever-increasing dependence on technology, participants generally liked the AID and gave very positive feedback on its use. In one participant's words: "*In the beginning I was sceptical, but now I'm very excited.*"

2.7. Discussion

At first sight, our measurements seem to disagree with the results of the user study by Nichols et al. [104], which showed that their participants took twice as long interacting with the actual physical interface even for those everyday tasks that required only "one or two button presses on the actual appliance". Nichols et al. attribute this superiority of the handheld approach to the "poor labeling, insufficient feedback, and overloading of some buttons with multiple functions" of the appliances under test. With respect to the AT&T phone they used for some of their experiments, Nichols et al. mention that "this phone has several buttons that can be pressed and held to activate another function. There is no text on the telephone to indicate this". Nichols et al. specifically addressed these drawbacks of the physical interface in the handheld implementation by providing an intuitive virtual interface, which explains the superiority of the handheld approach. In our experiments, the physical user interface on the four appliances is very intuitive and convenient for all four different everyday tasks. In the paper by Nichols et al. there is also little information on the actual tasks that were performed. The result is that we cannot compare the "one or two button" tasks mentioned by Nichols et al. with the everyday "one button" tasks we studied.

In our opinion, the value of the mobile phone to complete exceptional tasks stems from the shortcomings of the physical user interfaces and the corresponding manuals. User interface designers commonly have to deal with the conflicting constraints of cost, size, and usability. On the one hand, expensive high-resolution displays with many buttons help to build a user interface that is convenient and intuitive even for uncommon tasks. On the other hand, the cost and size restrictions typically limit the number of buttons available and the size and resolution of displays. For most appliances, the result is that interface designers are forced to implement uncommon tasks through a combination of buttons. The corresponding instructions are then listed in a complementary manual. This requires a significant effort from the user, however. The manual might not even be close-by, there is no direct link to the corresponding section in the manual, and the limited expressiveness of a manual makes the association of the printed instructions with the actual physical interface of the appliance non-trivial. The AID addresses all of these shortcomings, since it represents a cost-effective way to equip any appliance with a high-resolution display and a multitude of buttons. It thus enables user interface designers to address the conflicting constraints of usability, cost, and size associated with user interface design by leveraging the external mobile phone with its significant display and input capabilities.

The AID provides not only a high-resolution display and input capabilities, it also permits the personalization of the user interface due to the personal nature of the mobile phone. While designers usually have to develop an interface for the average user, the AID concept allows them to build a software interface that automatically adapts the language, adjusts the features available according to the capabilities of the user, and lists the history of recently performed tasks. The user interface designer would also benefit from the availability of well supported software development platforms, such as Java ME, Symbian, Android, and others. The long range communication capabilities of the mobile phone might also facilitate software maintenance, since the user interfaces can be updated remotely.

In our study, the AID software was preloaded on the mobile phone, and we simulated the appliance identification with a mobile phone. The rationale for simulating the identification was the limited display and computing capabilities of today's NFC-enabled mobile phones and the fact that our pilot study indicated that there was no discernable difference between real and simulated NFC action. As NFC technology becomes available in mobile phones with high-resolution displays, future user studies could incorporate the appliance identification and possibly also the downloading of the software. While we do not believe that this will have an impact on the findings of our study, it would make the overall application scenario even more realistic.

2.8. Summary

The goal of our study was to assess the benefits and limits of using mobile devices, such as PDAs or mobile phones, for appliance control. While the idea of a *universal remote control* is an appealing one, given the technical capabilities and prevalence of such devices, we questioned the sheer limitless uses that today's designers and researchers often envision for them. Hypothesizing that universal appliance controllers might be superior to traditional, physical appliance interfaces in *exceptional* situations only, but not for carrying out *everyday* tasks, we had our 23 test subjects perform a series of 18 tasks distributed among four appliances. By collecting quantitative measurements, we could confirm that our users were significantly faster when having to solve exceptional tasks with our AID (our prototypical universal appliance controller), but slower when performing everyday tasks. Our qualitative methods further confirmed that users would often prefer using the AID, but still liked the "natural interaction" with a device if a simple, straightforward task was to be solved and the tasks required them to be in the vicinity of the appliance anyway.

These findings seem to suggest that hybrid approaches that combine traditional, haptic user interfaces with extended user interfaces on a mobile device offer the best of both worlds. Users could continue to directly interact with appliances, which is both faster and more convenient in most everyday situations. However, in special situations where users would have to remember complex and clumsy sequences of pushing buttons or manipulating the appliance, it is much more intuitive to use a mobile device with its powerful and versatile user interface for interaction. The results thus suggest that the proliferation of mobile phones with high-resolution displays and short-range communication capabilities will enable appliance manufacturers to overcome the conflicting constraints on cost, size, and usability of the user interface, by leveraging the user's mobile phone.

3. Publishing and Discovering Services for Tagged Objects

3.1. Introduction

As we have seen above, research in ubiquitous computing has come up with a wide variety of applications for the paradigm of augmenting physical tagged products with digital information and services. Despite the diversity of the proposed scenarios, most of them assume monolithic, centrally administrated services — such as calling up an online dictionary when putting a bound dictionary on the desk [145], opening a product's web page [72], or launching an application-specific user interface [123].

However, the general availability of information tags and corresponding reader devices opens up the possibility for novel and innovative services that go beyond such isolated application domains. In order to leverage the potential of such applications, the underlying data that is linked to tagged objects should be available in a standardized form that eases exchange and use within different domains. While it is unrealistic to assume that a general-purpose data format can be designed, agrement on basic techniques to publish and exchange product-related information is needed for two main reasons: First, such an infrastructure is a necessary condition for the ability to find relevant information across application boundaries. Second, it removes the need for developers to design and implement afresh a custom solution for a generic problem.

Ideally, such an infrastructure would allow any party (e.g., manufacturers, consumer interest groups, governmental agencies, or even enthusiastic end-users) to dynamically add information and services to a specific product or product group, which could then be presented to and selected by users right when they scan a product.

In this chapter, we propose an *open lookup infrastructure* for tagged items, which allows

1. manufacturers, third parties, and individuals to publish product-specific resources (i.e., information and services), and

2. users to dynamically find and use these resources.

We begin by describing a set of envisioned scenarios and an analysis of the corresponding requirements for our lookup infrastructure in Section 3.2. The overall architecture of our system is presented in Section 3.4, and a prototypical implementation is reviewed in Section 3.5. Section 3.6 concludes with a discussion of three prototype applications that we built on top of our infrastructure, demonstrating the value and feasibility of our approach.

3.2. Application Scenarios

Augmenting physical products for end-user lookup is an attractive option, especially for manufacturers. A frozen food company could, e.g., differentiate its frozen spinach by providing an instant "recipe of the week" suggestion, which customers would be able to access simply by pointing their mobile phone at the product. At the same time, a consumer interest organization could provide background information about the product's health benefits, e.g., praising it for the organic fertilizer that was used for it, or maybe warning consumers of genetically modified ingredients. Another example scenario for end-user lookup could be a faulty appliance, such as a printer or a coffee maker, which would provide a diagnostic code over an integrated NFC interface. By touching the appliance with a mobile phone, users could pick up this diagnostic information and receive instructions on how to resolve the problem. Alternatively, a list of nearby repair centers could be displayed. For missing consumables (e.g., printing paper, coffee capsules, or filter units), third party stores could offer users one-click reordering options. Last but not least, tagged products could also provide machine-readable instructions for other appliances, which would, e.g., allow a microwave oven to prepare a frozen meal, or a washing machine to warn users when the wrong temperature for a certain garment is selected.

3.3. Requirements

Based on these scenarios, we can derive a number of high-level requirements for an infrastructure that should facilitate the discovery of resources (i.e., information and services) associated with a physical product.

Publication, search, and retrieval of resources In general, there can be many resources that are linked to a single tagged product at the same time. Our infrastructure must therefore provide mechanisms to store, structure, and find these resources. At the same time, an application might want to limit or focus the resources returned when looking up information and services associated with a given tagged product. Such search restrictions could be based on certain topics (e.g., "health aspects") or certain types of resources to look up (e.g., "expiration date"). In addition, the notion of context [29] should be supported as a search criterion. The inclusion of context allows an application to generate a meaningful ranking of relevant resources as well as to remove resources that are not considered relevant. Typical examples are the user's location when looking up a nearby repair center for a broken appliance or the appliance's error status to find the matching online troubleshooting guide.

Openness As the above example scenarios illustrate, a number of different and possibly dissenting stakeholders can have an interest in associating their resources with a given product. We therefore want to allow both product manufacturers and third parties (such as advocacy organizations, competitors, or individuals) to be able to publish resources for a particular product. Control over the system should not be concentrated, and a party should not rely on another party's cooperation when publishing a resource. Along the same lines, our system should also allow for the sharing of publicly available resources. We expect the general availability of resources and a supporting infrastructure to prompt the development of numerous innovative applications [79], just like it has been the case with web technology and the aggregation of diverse data sources in mashups.

Extensibility and flexibility In our application scenarios above, resources can be of very different types. Our infrastructure should thus not try to define a limited set of foreseen resource types and their uses. Rather than setting detailed standards that have a tendency of being either fairly rigid or overly complex, it should embrace evolution and change by defining "just enough" for participants to interoperate. Our infrastructure should therefore provide extension points that allow third parties, be it individuals or entire industries, to come up with their own resource types that can be shared through our system.

Interoperability Our infrastructure should be interoperable with existing information systems that already hold data. It should be easy to integrate such data, for example from an enterprise resource planning (ERP) system, and make them accessible as resources. In addition, it should also be interoperable with different numbering schemes, both current and future. There are already several product identifiers, such as the Electronic Product Code (EPC) [34], mainly used in the RFID domain, or the Global Trade Item Number (GTIN) [55], which is encoded in the majority of today's product barcodes. Future products may be known under several identities, such as an IPv6 address in addition to an EPC, or they may be labeled with entirely new identifiers. Our infrastructure should therefore be able to accommodate emerging numbering schemes.

Lightweight and secure architecture Most of the discussed applications run on either mobile devices or embedded systems. Due to the resource constraints typically found on these platforms, we need to employ lightweight protocols in our lookup infrastructure. The open nature of our system mandates the use of security mechanisms. For example, in certain applications users must be able to determine the authenticity of a party providing a resource.

3.4. Architecture

In this section, we discuss the architecture and the four core concepts of our lookup intrastructure (see Figure 3.1):

- resources and their descriptions
- resource repositories

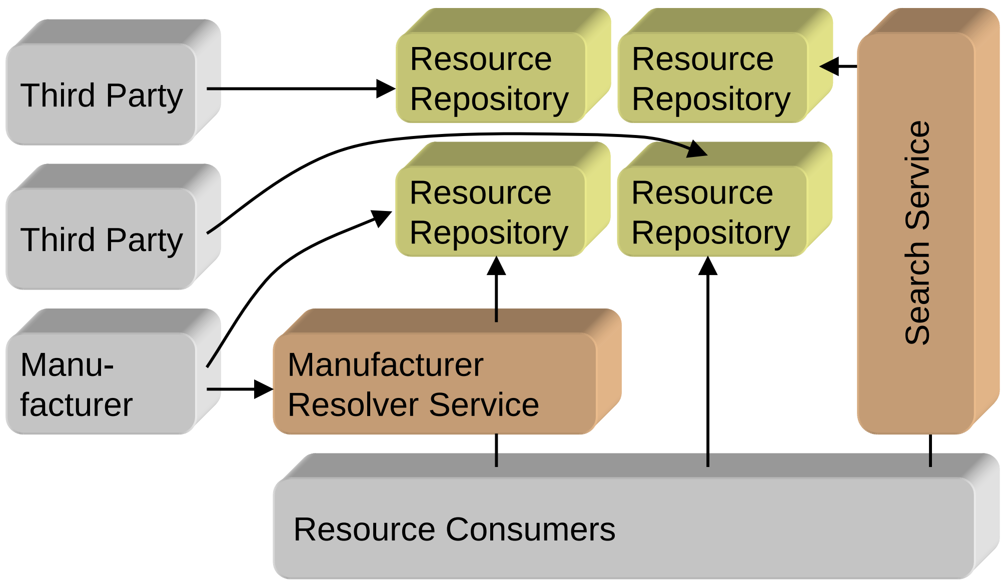


Figure 3.1.: *Open lookup infrastructure.* The center of our architecture are *resource repositories* containing *resource descriptions.* In order to find one or more repositories, given a particular product tag, mobile devices or stationary appliances either access a known repository (e.g., of their favorite consumer interest group), use the *manufacturer resolver service*, or query generic *search services.*

- a manufacturer resolver service
- search services

3.4.1. Resources and resource descriptions

Resources are at the core of our system. They offer information on, or services for, a physical product. Typical examples of resources range from a simple web site to complex web services. Resources can be provided by the original product manufacturer or any other party. Resource consumers can be product owners, business partners, or appliances. For every resource, a resource provider must create a *resource description* that specifies all the metadata that is needed to find and consume the information or service. The relation between resources and resource descriptions is illustrated in Figure 3.2.

Figure 3.3 shows an example of a resource description. A resource description includes the following main elements:

Resource ID The resource ID element is a pseudo-random value that serves as a globally unique identifier (GUID) for the resource.

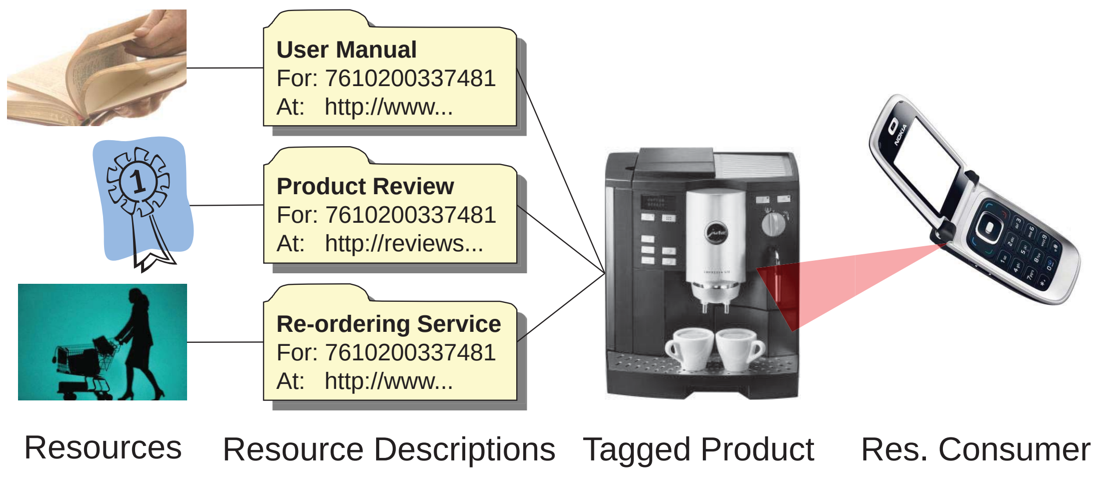


Figure 3.2.: Resource descriptions are used to establish a link between physical objects and resources.

Tag ID The tag ID element denotes the identifiers of those tags on physical products that a resource is associated with. The tag ID can specify a product at an item- or class-level. Different numbering schemes, such as EPC and GTIN (i.e., EAN/UPC), are supported. Note that a resource can carry several tag IDs at the same time and thus apply to several products. This can be helpful, e.g., when the same (or a similar) product is sold under different identifiers.

Profile The profile element can be used to express that the resource adheres to the syntax and semantics that are defined in a certain profile. Typically, a profile will be defined by an industry (e.g., in a standardization group). The food industry could, for example, specify in a profile how the expiration date of a product is to be represented in a resource. Profiles are essential in cases where a resource is not interpreted by humans, but processed by an appliance. Note that this element does not actually contain a syntactical or semantical description, but merely serves as an identifier for a format agreed upon by participants, similar to the *Content-Type* field in HTTP.

URL The URL element points to the actual resource (e.g., a web site). Alternatively, the resource can be stored directly in the *data* element if it is relatively small (e.g., a product's expiration date), which avoids an additional round trip. The syntax and semantics of the data available via either the *URL* or *data* element are

```
resource id:  f5f7305bf097af39c68b790d817d7889f788f222
     tag id:  urn:epc:id:sgtin:0614141.100734.1245
    profile:  http://foodindustry.org/profiles/expiration-date/
        url:  (empty)
       data:  2010-05-31
    context:  (empty)
      title:  Expiration date
description:  Expiration date for OrganicMilk, 1 liter
  signature:  (empty)
```

Figure 3.3.: *Example resource description.* Descriptions can be expressed in various formats, e.g., XML or even binary, depending on the particular communication and storage needs of a product (example given in an abstract format).

defined by the resource's profile as indicated in the *profile* element.

Context If specified, the context field defines in which situations the resource is considered relevant. In order to enable interoperability, we predefine the following context elements that can be used to restrict a resource's applicability:

- time (date, time, weekday)
- location (coordinates, city, country)
- status of the appliance the user interacts with (expressed as a simple string)

Note that this list is easily extensible by resource providers. Exact values, value ranges, and regular expressions are supported for each context element.

Title The title element assigns the resource a short title in natural language.

Desciption The description element describes the resource in natural language.

Signature The resource provider can digitally sign the resource description using the optional *signature* element.

Figure 3.3 shows an example of a resource description, in this case describing the expiration date of a particular bottle of milk. Note that the example is given in an abstract format, which in practice can be instantiated in a number of formats, such as XML or even binary form, depending on the particular use case. Also, resource descriptions for food products might equally well be entire data sets (e.g., expiration date, allergy information, country of origin) instead of just a single data item (e.g., expiration date) as in the above example — this can be standardized as needed by the various standard bodies.

3.4.2. Resource repository

The resource repository is responsible for storing resource descriptions and making them available to resource consumers. Resource repositories can be deployed by any party interested in offering resources, such as a manufacturer or an advocacy group. In this way, a single resource repository typically contains the descriptions of resources that are thematically related. Operators can flexibly configure access restrictions to their resource repositories. For example, a manufacturer will in most cases run a read-only repository, while a community-operated product reviews repository might allow anyone to add or even update resource descriptions (very much like a wiki system). The same applies to the querying side, where a consumer reviews publisher might limit access to its repository to paying customers only.

The three basic operations offered by a repository are `RegisterResource` to publish a resource description, `RemoveResource` to delete a published description, and `LookupResource`, which returns the descriptions of those resources matching the query conditions provided by the caller. A query can consist of up to four elements:

Tag ID The tag ID element must be provided to denote the product for which resources are looked up. A lookup can be performed at both class- and item-level.

Profile This element indicates that only resources adhering to the given profile should be retrieved.

Search term A search term element can be specified to restrict the resulting resources based on their textual description.

Context Using the context element, the caller can specify an arbitrary number of context values. Each value must be marked as either a *hint* (favoring resources with a matching context element) or a *requirement* (excluding resources with no matching context element).

```
     tag id:  urn:epc:id:sgtin:0652642.800031.400
    profile:  http://appliances.org/troubleshooting-hints/
search term:  (empty)
    context:  status=E683[hint]
```

Figure 3.4.: *Example lookup request sent to a resource repository.* Based on the printer's status that was retrieved via its NFC interface, a user could query directly for information pertaining to the particular printer in the context “status=E683”.

A typical lookup request is depicted in Figure 3.4. It shows a request as it could, for example, be sent to a printer manufacturer's resource repository in order to obtain troubleshooting instructions when the printer is in a malfunctioning state. The printer's status code is read by the mobile phone's NFC module and used as context information to narrow down the lookup.

A resource repository can also be configured to allow user feedback on resources. The incorporation of feedback allows community-based applications where the quality of content is controlled by users submitting confidence values for resources. At the moment, only a `SendBinaryFeedback` operation is provided, which can be called by users to express their approval or disapproval of a resource. The order in which resource descriptions are returned by the repository depends on these ratings. Finally, the resource repository can be configured to synthesize resource descriptions of a specific profile through custom-built *wrappers*. Wrappers can be used to integrate existing information systems, such as an ERP, into the lookup infrastructure.

Note that resource repositories are in principle no different from traditional web servers. Therefore, the same well-established mechanisms for achieving security, reliability, and scalability can be used. For example, a repository could be replicated and made accessible through a load balancer that routes traffic according to the individual repositories' availability and load.

3.4.3. Manufacturer resolver service and search service

In order to make use of resource descriptions, users must be able to locate the resource repositories containing them. This is the task of the *manufacturer resolver service* and *search services*. They connect a product tag ID to a resource repository where resource descriptions pertaining to the product can be found.

The use cases in which the various deployed resource repositories are accessed by potential resource consumers can be divided into four groups:

1. In the first group, only the product manufacturer's repository is of interest. An example for such a case is a washing machine that checks the handling instructions of every garment put into it.

2. In the second group, there is a single repository that is used for every lookup. An example for this case is a service that aggregates prices from several stores and lets a user check whether the physical product at hand could be obtained from a cheaper source.

3. In the third group, a lookup is performed in several repositories at the same time. An example for such a case is a browser application that lets users specify a number of repositories operated by interest groups (e.g., environmental, political, etc.) they care about. The browser would then, for example, display all reviews regarding a product that can be found in the repositories relevant to the user's interests. We envision repository directories similar to the Dmoz Open Directory Project[1] from which users can pick the repositories they find interesting.

4. In the fourth group, a user wants to search all repositories for resources associated with a given product. This case comes into play when no relevant resources can be found in the repositories the user has registered in his or her browser. In this case, the consumer would simply query his or her favorite search service for relevant repositories.

It is clear from these considerations that the architecture needs to include both a *manufacturer resolver service* that links a tag ID to the manufacturer's resource repository as well as a *search service* to find resources across the boundaries of single repositories.

Why is there only a resolver service for manufacturers? Why not for distributors, vendors, or consumer interest groups? After all, the example scenarios in Section 3.2 above illustrated that a wide variety of parties might want to offer their descriptions to consumers, each for an

[1]`www.dmoz.org`

equally valid reason. The question of who gets to supply information to a product, i.e., who gets to "define" its properties, is actually highly political. The open lookup infrastructure uses a pragmatic approach, inspired both by technology and legal realities. Manufacturers already play a special role in the life of a product. They are responsible for its safety, they supply manuals, organize warranty and repair, and often also handle its recycling. In many scenarios, manufacturers thus will be legally the main, if not the only, authoritative source for information. From a technical point of view, manufacturers are also much easier to localize, given the product's identifier. This is because the current EPC standard (and, to some extent, also the EAN/UPC standards) contain special mechanisms to quickly identify a product's manufacturer from an EPC or GTIN code. Our manufacturer resolver service makes use of this mechanism (see Section 3.5.2 below for details), thus ensuring that users can always locate the repository of a product's manufacturer.

All other information and service providers are harder to identify and find. While one could conceive a central registry where all repositories would be registered, this would violate our *openness* and *extensibility* principles set forth in Section 3.2. Instead, we decided to complement our manufacturer resolver with an orthogonal, decentralized, search-based approach, building on existing web search technology. Just as today's web spiders and robots, specific resource search services would crawl repositories, create an index, and answer search queries (see Figure 3.5). A query passed to a search service's `Search` operation consists of the same four elements (tag ID, profile, search term, context – see above) as a `LookupResource` request sent to a single resource repository.[2] Of course, users can also directly access repositories (e.g., of their favorite product review magazine) by manually entering its address, by receiving the address via Bluetooth or SMS, or by finding it in a directory of resource repositories.

3.4.4. Deployment and use

How would these architectural parts be used to deploy and/or make use of individual product descriptions? This depends on the individual stakeholder.

[2]While this mechanism could in principle be also applied to the manufacturer's repository, thus eliminating the need for a special manufacturer resolver, we decided to make use of existing resolution mechanisms in order to guarantee users that at least the manufacturer information can be located.

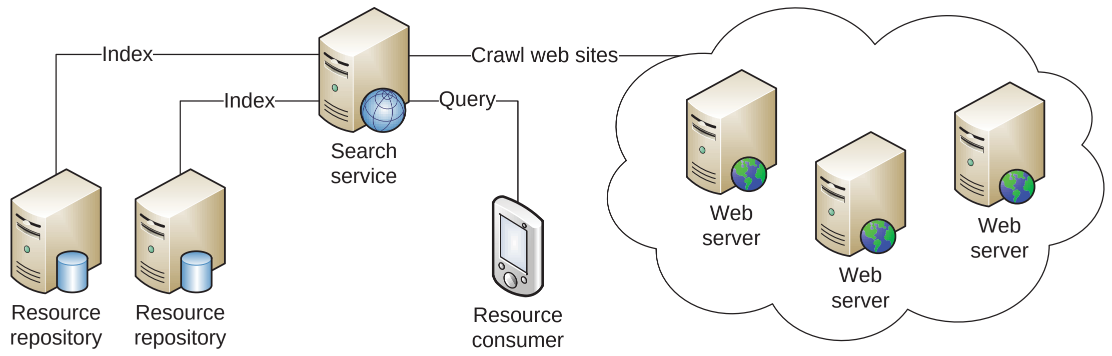


Figure 3.5.: Search service and its sources, including resource repositories and traditional web sites.

A *manufacturer* would begin with setting up a public, read-only resource repository, e.g., using an add-in to a standard web server. It would then create resources for each of its products – either informational resources such as web pages or user manuals, or service resources, such as a recipe service or a diagnostics program – and prepare corresponding resource descriptions for each of these resources. These would be entered into its resource repository, which in turn would be registered with the manufacturer resolver service.[3]

A *third party* wishing to provide information for a certain product (e.g., an advocacy organization or even a government agency) would start out similarly. After setting up a repository, creating a number of resources and publishing their descriptions in the repository, however, a third party would need to advertise this repository to potential users, as it cannot make use of the manufacturer resolver service. Instead, it would register its resource repository with a search service or repository directory (similar to the early Yahoo! or the Dmoz Open Directory Project), and/or communicate its repository URL to end-users through traditional advertising (e.g., TV, SMS, and print media).

Without any special configuration, *end-users* can always contact the manufacturer's resource repository, which can be found via the manufacturer resolver service, in order to retrieve a list of "official" resources offered for a product. Similarly, they can use a search service to find resources available from third parties that have registered their repository with the search service. Alternatively, they can manually configure resource repositories that they find particularly interesting, using the above mentioned out-of-band advertising mechanisms.

As with the World Wide Web, the cost of running our infrastructure is

[3]See Section 3.5.2 for details on how the manufacturer's resolver service is registered.

borne by those publishing resources. Parties interested in participating must either set up their own resource repository (i.e., a dedicated server operating on a 24/7 basis), or find someone to do so on their behalf (e.g., a hosting company). Just like the web, our repository infrastructure can be built gradually and without central coordination.

Since anyone can publish arbitrary resources, data quality will become an issue. Until sophisticated search engines that can provide ranked results are available, we expect that word-of-mouth recommendation and independent editorial review (e.g., popular press) will lead to the emergence of a set of resource repositories that are known to provide quality content. Just as it has become standard with web sites today, manufacturers and third parties will eventually run and advertise their repositories in both print and electronic media, treating them as yet another means for differentiating their products and services.

3.5. Implementation

Based on the concepts described above, we implemented a prototype of our resource lookup infrastructure. For each of its three building blocks, the implementation is reviewed in this section.

3.5.1. Resource repository

Resource repositories are implemented using Java Servlets using the Spring framework[4] and a relational database for resource, feedback, and user management. Fulltext search capabilities are implemented using the Apache Lucene[5] search engine. The implementation provides bindings to SOAP, XML-RPC, and REST [41]. Optionally, TLS can be used for increased security.

3.5.2. Manufacturer resolver service

Resolving the manufacturer's resource repository is implemented using the Object Naming Service (ONS) [33]. ONS is a global infrastructure that is used as part of EPCglobal's EPC Network to find the EPCIS repository[6] of a product's manufacturer. It resolves a prod-

[4]`www.springsource.org`

[5]`http://lucene.apache.org`

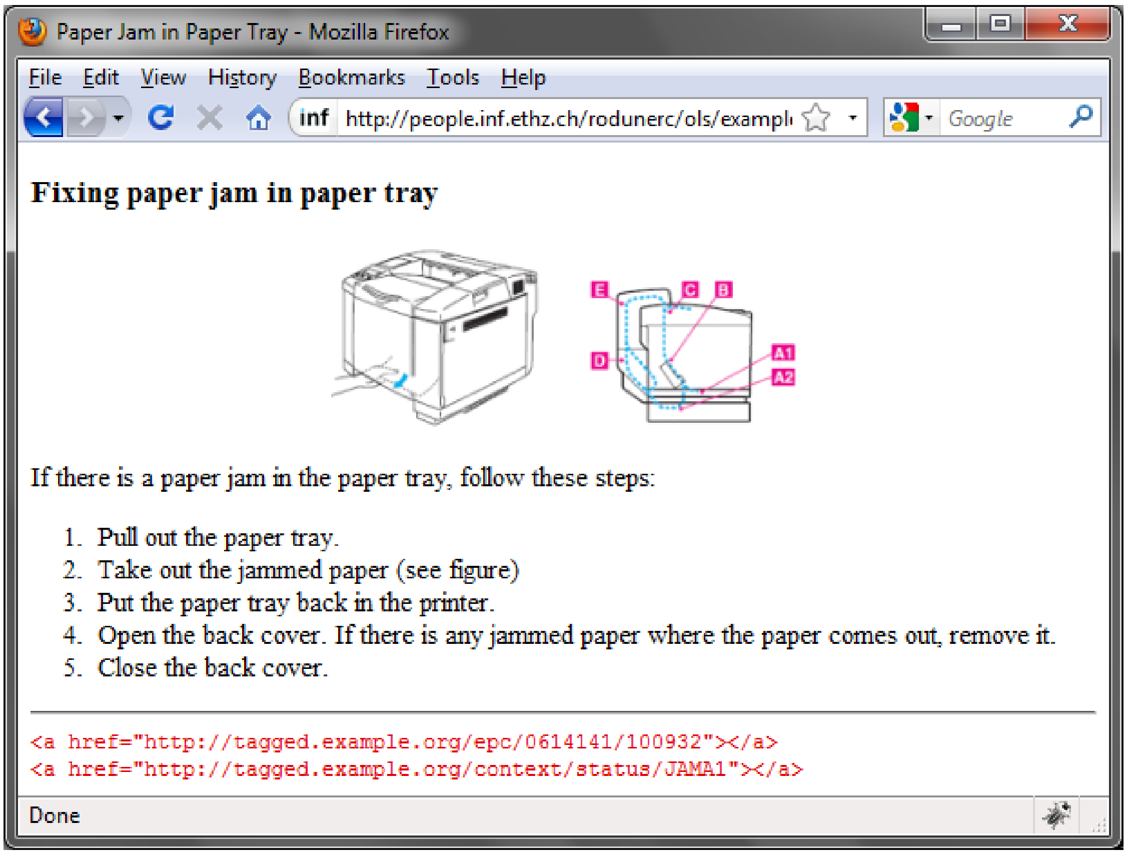
[6]EPCIS stands for *EPC Information Services* and is an integral part of the EPC Network. EPCIS holds logistical information on a product in the EPC-enabled industrial supply chain.

uct's identifier (its EPC number) to a URL pointing to the corresponding EPCIS repository by leveraging the existing Domain Name System (DNS) infrastructure. The basic principle of ONS is to append ".sgtin.id.onsepc.com" to the EPC's string representation with the item reference stripped. Using the standard DNS infrastructure, the resulting domain name (e.g., `000024.0614141.sgtin.id.onsepc.com`) is then queried for "NAPTR" records (a type of record as defined by the DNS specification [87]), which contain the URL to the manufacturer. We use a custom value (`EPC+ResRep`) in the `service` field of the NAPTR record in order to distinguish our URLs pointing to the manufacturer's resource repository from other data in the ONS (typically URLs pointing to an EPCIS repository).

3.5.3. Search service

We believe that indexing of resource repositories is a task that could be best done by already existing web search services. In our prototype system, we developed a simple search service based on the Apache Lucene search engine. Our search service crawls all registered resource repositories, creates an index, and can be queried using the `Search` operation.

In addition to this, we extended our search service implementation beyond crawling resource repositories. The internet is full of standard web pages containing information that pertain to physical products. Such information ranges from product reviews to user guides and blog entries. If we consider such standard web pages as potential resources linked to physical products, we can easily build a search service for these particular resources. Similarly to the Technorati blog search service, we use an empty "anchor" tag (i.e., an `<a/>` HTML element) to mark a web page as being a resource belonging to a certain physical product. A weblog author could for example link a posting to a physical product with EAN number 7610200337481 by including the element `<a href="http://tagged.example.org/gtin/7610200337481"/>` as a *marking* into the HTML source code. Note how this link does not enclose any text, which is how traditional hyperlinks work. Instead, this singular anchor indicates that the entire page applies to the referenced resource while remaining invisible to users viewing the page in a traditional web browser. Similarly, an author can indicate the context in which the web page is considered relevant. An el-


Paper Jam in Paper Tray - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://people.inf.ethz.ch/rodunerc/ols/exampl
Google
E
C
B
D
A1
A2
Done

Fixing paper jam in paper tray

If there is a paper jam in the paper tray, follow these steps:

1. Pull out the paper tray.
2. Take out the jammed paper (see figure)
3. Put the paper tray back in the printer.
4. Open the back cover. If there is any jammed paper where the paper comes out, remove it.
5. Close the back cover.

```
<a href="http://tagged.example.org/epc/0614141/100932"></a>
<a href="http://tagged.example.org/context/status/JAMA1"></a>
```

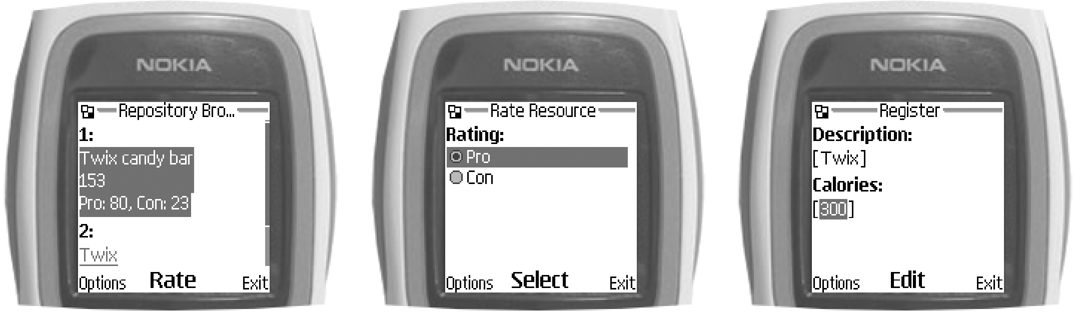
Figure 3.6.: *Web page with marking for search service.* By including the two `<a>` elements (printed in red), the author of this web page indicates that it pertains to tagged products with the EPC manager set to 0614141, the EPC class set to 100932 and a device status code of "JAMA1". In other words, the page is relevant to a particular printer model in a particular device status. Note that the `<a>` tags are not usually visible and are printed for illustrative purposes only.

ement of `<a href="http://tagged.example.org/context/status/JAMA1"/>` would indicate that the containing web page applies to situations where the tagged object's error status is "JAMA1". Figure 3.6 illustrates the use of such markings in traditional web pages.

As most search engines support a *link* operator to find all web pages linking to a given URL, it is possible to leverage these systems to easily find pages marked with such an `<a/>` element. Our original intention was to implement the search service around one of the large internet search engines. However, as this turned out not to work reliably, we again used Apache Lucene as the underlying search technology. When the search service receives a `Search` request, it internally queries the Lucene search engine, converts the search results into resource descriptions with the *profile* element set to "webpage" and the *URL* element set to the respective web page's address, and returns these resource descriptions to the caller.

Depending on the client's request, matching resource descriptions found in resource repositories and synthesized from web pages are returned either separately or in aggregated form. Our search service implementation provides bindings to both XML-RPC and REST.

3.6. Prototype Applications



1:
Twix candy bar
153
Pro: 80, Con: 23
2:
Twix

Rating:
○ Pro
○ Con

Description:
[Twix]
Calories:
[300]

(a) Results overview (b) Rate (c) Add resource

Figure 3.7.: *"Calorie Tracker" application.* An example for a community-built and -maintained product repository.

To demonstrate the use of the open lookup infrastructure by developers of applications around tagged products as well as to validate our architectural design choices, we built three prototype applications. All demonstrators were implemented as Java MIDlets on a Nokia 3220 mobile phone. The MIDlets use the REST binding to connect to both the resource repositories and the search service. XML parsing of service responses is implemented using kXML[7], a lightweight parser for J2ME with minimal memory footprint. Our demonstrators rely on the Nokia 3220's integrated NFC reader, even though conventional EAN/UPC barcode symbols could be equally used as tagging technology.

3.6.1. Calorie tracker

The first demonstrator allows users to track their daily calorie intake (see Figure 3.7). Calorie information on products is fetched from a user-extendable resource repository. The application demonstrates the possibility of a community-built and -maintained resource repository, by creating new resources and adding feedback to them directly on

[7]`http://kxml.sourceforge.net`

a mobile phone. To ensure basic quality control, we borrow a concept from community websites and let users approve or disapprove of resources created by other users. For every resource, the number of positive and negative votes is recorded and taken into account when resources are ranked in response to a query. If there are no entries for a product, or if a user does not agree with any of the returned values, a new resource can be created. When a user touches a product with the NFC phone, a `LookupResource` request with the acquired tag ID is performed on the "calories repository". The result contains a list of resource descriptions, consisting of the textual description, the calorie number, and feedback, as partly shown in Figure 3.7(a). While browsing through the results, the user has the possibility to rate a result. Figure 3.7(b) shows the form for entering a rating for a resource. If none of the suggestions are correct, the user can add a new resource as shown in Figure 3.7(c).

Table 3.1.: *Resource repository queries.* Three examples for a shopping assistant (see Section 3.6.2), trying to find information pertaining to an identified product.

Repository	`LookupResource` elements
manufacturer	`tagid=urn:epc:id:sgtin:0614141.749126.8372, profile=allergy`
price information	`tagid=urn:epc:id:sgtin:0614141.749126.8372, profile=price`
env. information	`tagid=urn:epc:id:sgtin:0614141.749126.8372, profile=review`

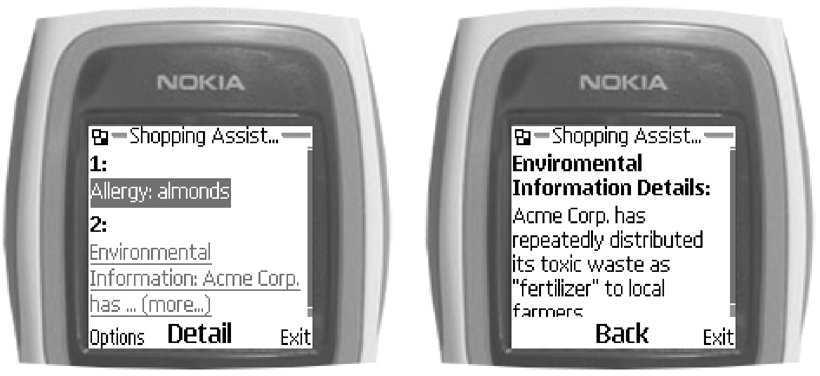
3.6.2. Shopping assistant

A second example application provides users with background information on products. Upon scanning a tagged product, the "shopping assistant" contacts three resource repositories: First, the manufacturer to obtain allergy information according to the "allergy" profile that we assume has been defined by the food industry. Second, a repository implementing a wrapper to the product's price information at Amazon.com. Third, a repository offering information on environmental issues of a given product. Based on the resources obtained from these repositories, the assistant informs the user if the product conflicts with his or her allergy profile, if it is available from Amazon and for what

```
<resDescriptions repository="http://repos.allergy.org/">
  <item resId="b5fe3a5bf077af32c68b790d817d7339f724f209">
    <profile>allergy</profile>
    <title>Allergy Information</title>
    <data><almonds/></data>
  </item>
</resDescriptions>

<resDescriptions repository="http://repos.envprot.org/">
  <item resId="73cd125bf097af69c64b790d817d7899f788ffa7">
    <profile>review</profile>
    <title>Environmental Information</title>
    <data>Acme Crop. has repeatedly distributed its toxic waste...</data>
  </item>
</resDescriptions>
```

Figure 3.8.: *Responses from resource repositories.* These (abbreviated) replies illustrate possible responses to the queries shown in Table 3.1.



Shopping Assist...
1:
Allergy: almonds
2:
Environmental Information: Acme Corp. has ... (more...)
Options Detail Exit

Shopping Assist...
Enviromental Information Details:
Acme Corp. has repeatedly distributed its toxic waste as "fertilizer" to local farmers
Back Exit

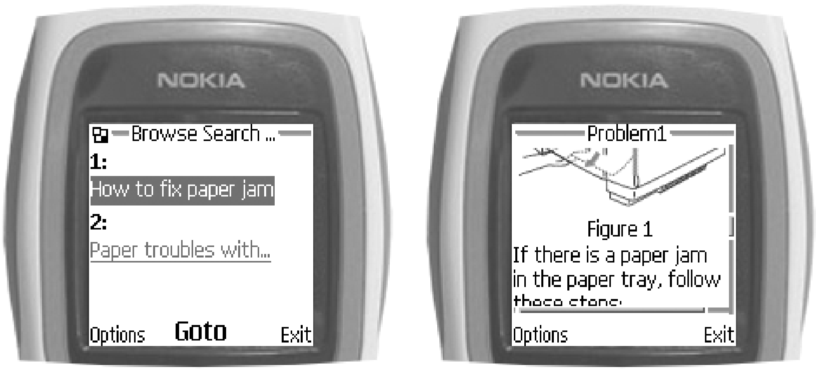
(a) Overview (b) Details

Figure 3.9.: “Shopping assistant” application.

price, and if there are any environmental issues known about it. Table 3.1 shows queries for an example product sent to the three repositories, while Figure 3.8 illustrates two received responses. All results are aggregated and displayed as shown in Figures 3.9(a) and 3.9(b). The Amazon book price resources are automatically created by a custom wrapper that leverages the Amazon Web Services to fetch the current price of books.

3.6.3. Appliance support

Our last application uses context in the form of a status code obtained from a malfunctioning appliance, such as a printer, to find information that can help solve the problem. We use the search service to locate web pages, blog entries, or other sources of information that are



How to fix paper jam

Paper troubles with...

If there is a paper jam in the paper tray, follow these steps:

(a) Printer help (b) Help details

Figure 3.10.: “Appliance support” application.

marked as relevant to the product at hand in the status encountered. Figure 3.10(a) shows an overview of the results found for a printer in a certain status. By selecting “Goto”, the user can launch the device’s web browser and open the web page (Figure 3.10(b)).

3.7. Related Work

Resource discovery, i.e., finding resources by specifying a set of desired attributes in a distributed environment, has been an active field of research. However, many of the protocols in this domain, such as Jini [138, 144], UPnP [142], SLP [56], and Konark [58], focus on ad-hoc networks and, often relying on multicast, cannot easily be used outside a single administrative community. Even though some (notably SLP) scale better, these protocols are not suited for scaling to the extent required by the application scenarios outlined above, which span many administrative communities (i.e., the internet).

Another widely known protocol for automatic service discovery is UDDI [141], which is used in the Web Services domain. Our approach differs from UDDI and the earlier, but similar E-Speak [69, 68] protocols. These systems are strongly oriented towards enterprise applications and should in principle scale well. They are part of a comprehensive framework that offers a rich feature set, including, for example, replication, access control mechanisms, and comprehensive matchmaking capabilities. However, they tend to be fairly complex and incur a significant overhead with respect to both communication protocols and software development. They are thus not ideal candidates for the scenarios we have in mind, where lookup operations should be lightweight and normally result in the exchange of small bits of information only.

Several systems have aimed to address the shortcomings of the above procotols in a ubiquitous computing environment:

Friday et al. [46] propose a solution that extends the above protocols by supporting arbitrary metadata (according to an application's needs) and integrating the various existing protocols. In a heterogeneous environment that relies on different discovery standards, they deploy a "service interpreter" for every protocol used. An interpreter collects all service announcements of the respective discovery protocol. Whenever an announcement is observed, it is converted into an SQL "INSERT" statement that is submitted to a database. To find a service, a client sends an SQL "SELECT" query to the database that returns the matching services. This architecture has two main advantages: First, a client can find a resource irrespective of the underlying discovery protocol that is used to publish it. Second, custom attributes to describe services can be supported by simply storing them in the SQL database. This architecture is related to our open lookup infrastructure because both systems share the idea of a lightweight system and support for arbitrary meta-data. However the main advantage of Friday et al.'s approach is the integration of different protocols. Several other aspects that are part of our open lookup infrastructure, such as a possible data model to allow for custom extensions, context, or discovery across database boundaries, are not discussed in their work.

Castelli et al. [20] present a data model and infrastructure for the management of "contextual information" on the real world. Their data model is based on "knowledge atoms", four-field tuples that describe the *who*, *what*, *where*, and *when* of facts inferred from sensors, such as RFID readers. These knowledge atoms are stored in tuple spaces that can be local or remote. As a typical example application, the visualization of tuples in Google Maps is described. The idea of using several tuple spaces or repositories to store contextual information resembles the one used by our infrastructure. Also, both systems explicitly support context-based lookup operations. However, Castelli et al. do not discuss how such an infrastructure would actually be deployed and scaled. Also, their work is strongly focused on RFID read *events* (expressed by the "who, what, where, when" dimensions of their data model) and is not suitable for associating other *resources* with a tag.

Schmitt et al. [134] discuss an "Open Object Information Infrastructure (OOII)", which shares the same vision as our open lookup infrastructure: To provide an environment where innovative services, building around information on smart objects, can thrive. However, their focus seems to lie on how raw RFID read events can be transformed into semantically enriched events. While they explore the benefits of such an infrastructure, their work is at a fairly high level of generality and does not come up with a specific system or architecture proposal.

What is common to the three projects by Friday et al., Castelli et al., and Schmitt et al. is that they are designed for an environment that is rather dynamic. They are concerned with the collection, storage, and distribution of automatically generated data. Often, the data source is a sensor whose output needs to be interpreted before further processing. This can result in a relatively high data volume that must be absorbed by these systems in a short time. These characteristics make them different from the open lookup infrastructure. Our infrastructure is concerned with the management of fairly static resources, such as reviews, prices, or instructions. Many of them are generated by humans, rather than a sensor, which makes updates significantly less frequent. This allows us to rely on search services that crawl repositories only occasionally. While this means that timely updates are not always available, it is an acceptable trade-off in the scenarios we have in mind.

Another related project is the Context File System (CFS) developed by Hess and Campbell [59] in the context of the Gaia middleware [125]. CFS allows for the storage of data in such a way that relevant material can be located by users and applications based on context information, user preferences, and device characteristics. It is similar to our open lookup infrastructure in that resources can be organized and returned based on the context in which they are relevant. Like our system, it is also extensible and supports user-defined context types. However, Gaia's CFS is monolithic. It is deployed by a single organization and does not allow content providers to retain full control over their resources. Also, CFS does not support result collections in response to a query to be ranked according to a user's preferences. This is particularly important, since we expect result collections to be potentially large. Finally, CFS has no provisions to include external data.

INS/Twine by Balazinska et al. [7] is a resource discovery infrastructure for ubiquitous computing applications that leverages distributed hash tables as its underlying data structure. It therefore shows good scalability properties and is well suited for large networks. However, it gives resource providers no control over where their resources are stored, which would be a problem for tagged product resolving, as it gives resource providers no control over service quality and cost sharing.

In both the Context File System and INS/Twine, the publisher of a resource has to give up ultimate control over it. It is stored in and advertised by a system that is operated by another party. In addition, it is not possible for resource publishers to switch to another supplier in case they are not satisfied with the service level that is provided. In the commercial scenarios that we have in mind, such properties are crucial. The open lookup infrastructure thus ensures that resource publishers have a choice of a) either running their own resource repositories and ultimately retaining control over their content, or b) delegating the task to any of the third parties offering such services.

IBM Research's Project Celadon [86] is a framework for providing shoppers with relevant commercial information as they move around in a public space, such as a shopping mall. In the scenario addressed, it resembles the Shopper's Eye application proposed by Fano [39]. Celadon allows the environment as a whole, applications in this environment, or users themselves to offer services. Users are the core entities of the framework. Their location is monitored by the system, they can take logical roles (such as "shopper", "premium shopper", or "employee"), and they can receive offers made by the environment's applications. Celadon shares with our open lookup infrastructure a similar data model as well as a fairly simple and lightweight API built upon web standards. However, tagged objects (i.e., physical products) play no role in the Celadon framework, and it is therefore not possible to associate services with them. As interaction in Celadon cannot be initiated by reading a physical object's tag, it relies on DNS multicast for bootstrapping before information can be sent to a user entering a new environment. Overall, Celadon's focus is more on location tracking, matching of customers' interests with services offered by shops, as well as leveraging the environment's infrastructure (e.g., large public displays).

Very much related to the open lookup infrastructure is EPCglobal's EPC Network [139, 35], and in particular the EPC Information Services (EPCIS) [32]. The EPC Network is perhaps the most prominent architecture that addresses the need to collect and share product-related information based on auto-id technology. However, the EPC Network was designed with logistics and enterprise applications in mind. It does not currently provide any means for end-users to access a product's data trail. Like UDDI, it makes heavy use of Web Services standards, which are not an ideal fit for the scenarios addressed by the open lookup infrastructure (see above). Most of all, however, the focus of EPCIS lies on product *event data* rather than *master data*. In the EPCglobal framework, event data grows as business is transacted (e.g., when a shipment arrives), whereas master data does not grow and is not tied to a specific moment in time. Of these two categories, the information and services discussed above fall into the latter one. As EPCIS only provides an API for querying, but not for collecting master data, it is different from the open lookup infrastructure.

3.8. Summary

The idea of linking information and services with physical objects is a powerful concept, especially when we are able to augment millions of everyday products with such resources. Realizing the vision of every product being augmentable raises the question of how interested parties can flexibly associate information and services with a product. This chapter addressed this issue by presenting the concept and architecture of an *open lookup infrastructure* that fulfills the requirements derived from a range of example application scenarios. We validated the infrastructure by implementing its key components prototypically. We also implemented three demonstrator applications to illustrate how it facilitates the development of novel applications involving digitally augmented, tagged products.

In summary, the open lookup infrastructure offers four key benefits to the various stakeholders involved. Firstly, it allows users to find out what information and services are available for a physical product. Secondly, it gives resource providers access to potential consumers. Thirdly, it enables manufacturers to increase the value of their products by adding information and services to them. And finally, it provides application developers with concepts and services that facilitate the implementation of novel applications.

4. Facilitating Service Development – A Browser for the Internet of Things

As we saw in Chapter 2, personal mobile devices can serve as an intuitive tool for end-users when controlling physical appliances. In Chapter 3, an infrastructure was presented that can be used to link all kinds of information and services to physical objects that are equipped with some sort of tag, such as a barcode or an RFID label. In this chapter, we will present BIT, a *browser for the Internet of Things.* BIT pursues two objectives. On the one hand, BIT aims to be a "single point of interaction" for users by integrating both appliance control and the various other services that can be linked to a tagged object. On the other hand, BIT provides a software framework that considerably facilitates the development of mobile phone-based services for tagged objects.

This chapter is structured as follows: We will begin by briefly highlighting some earlier ideas of how users can retrieve information and services from physical products. In Section 4.2, we analyze the problems that developers face when creating mobile services for tagged objects today. Section 4.3 present a scenario that illustrates how BIT is used, and from which we derive our requirements in Section 4.4. In Section 4.5, we discuss the implications of these requirements for the design of our system. Based on these findings, Section 4.6 details the core concepts underpinning our framework. Section 4.7 shows how these concepts and our framework are applied to service development. In Section 4.8, we present the architecture and implementation of a BIT prototype that provides support for our framework. In Section 4.9, we demonstrate the applicability of our system and its practical value for developers by implementing nine diverse services. After a discussion of its strengths and weaknesses in comparison with traditional development techniques, we conclude by contrasting BIT with other related projects in Section 4.10.

4.1. Introduction

The idea of a personal and portable device that would allow its users to retrieve background information on consumer products has been discussed for a while. An early example is the "personal shopping assistant" [6]. It was proposed in 1994 as a mobile device of the "size of a walkman" with an integrated barcode scanner that could be carried around a retail store by shoppers. It would show product information and also keep a running total of all items purchased. The Pocket BargainFinder [14] later implemented some of these ideas, using a PDA combined with a barcode scanner as its hardware platform. The system allowed users to scan the UPC codes of books and would subsequently check prices and product availability from online resellers.

Due to the obvious commercial potential, the use of mobile devices to provide product-related services was later explored by several other projects, often seeing "m-commerce" as an extension of the e-commerce boom at the time.

Safeway UK and IBM tried out an application for PDAs that would assist shoppers in creating orders for grocery products while at home [11]. These orders would then be transmitted to a store, which would prepare the order for later pickup. Shopping recommendations based on data mining were later added [78]. In another project [100], such applications were investigated from a usability perspective. In a user study taking place in an actual retail store, a number of different possible use cases for mobile devices were analyzed. The system used in the trial, running on a PDA, would let users prepare their shopping lists, display the most efficient route through the store, and offer recipes or coupons. While these projects were built around physical products and mobile PDAs, none of them made use of electronic labels to recognize products.

In the context of the proliferation of RFID into the supply chain, the technology's potential to enable interaction with products in retail store environments was also studied. An example of such a project is MyGrocer [127, 74], which analyzed the use of a tablet PC to offer self-checkout and product information in a brick and mortar store.

These projects have all explored the potential value that mobile devices can offer for applications around consumer products. Some of them have used tagging technologies, others have not. However, all these systems have been standalone applications. None of these projects have investigated how the many application ideas can be integrated in a consistent, unified framework. Neither have they looked into the specific needs that arise when all these applications are to run on the end-user's own mobile device that cannot be assumed to be available for exclusive use by these services.

4.2. Limitations Today

While the scenarios outlined above are likely to be appealing to most end-users, developers of such services face a number of challenges due to the limitations of the technology that is available today.

Application development for mobile phones is still a cumbersome process [53]. Toolchain and framework support is generally poor. At the same time, the applications that implement the services described above are often very small and do not offer extensive functionality. Acquiring deep skills in mobile application development and going through the process of setting up a full-fledged development environment is hardly justified. Moreover, many applications are provided by product manufacturers with no expertise in software development. While application development can certainly be outsourced, this is not the preferred way, since it makes solutions static and hard to adapt as the underlying physical products evolve. Manufacturers need a way to quickly deploy new services as they, e.g., launch new marketing campaigns. The same applies, of course, to traditional web sites accompanying a physical product. Unlike mobile application development, however, the programming of an interactive web site is a considerably simpler task, which requires skills that are much easier to acquire.

The problem is further complicated by the fact that today's mobile phone market is still fragmented into a number of different, incompatible platforms, such as Java ME, Symbian, iPhone, Android, Palm webOS, and others. Every mobile application thus needs to be ported to all major platforms. As an obvious answer to this dilemma, manufacturers may opt for web-based solutions. The downside of this approach is that a mix of HTML, JavaScript, and server-side applications does not enable developers to access phone-specific hardware, such as barcode or RFID readers as well as GPS receivers. Another problem arises from the relatively high latency in today's mobile networks. Applications based on web technology require frequent request-response cycles that are triggered by simple user input. Given the latency commonly observed, this has a negative impact on user experience. Finally, there are still some places without network coverage, such as subway stations. Even without network coverage, a physical object may offer information or services directly to the mobile phone through, e.g., NFC technology or Bluetooth.

From a usability perspective, another challenge stems from the necessity to integrate the various services into a single interaction framework. While services can easily be created and deployed as independent applications, this results in the user's phone being littered with a large number of small programs. Worse yet, the execution of these applications is not automatically coordinated. In order to use a service, the user first needs to manually start the corresponding application before a product can be scanned. Since many objects will not be associated with a given service, scanning will not render a result in many instances, which will discourage users from further interaction. A user's need to simply see "everything my favorite services offer for this product" cannot be addressed in current architectures. A user would have to manually launch one application after another to check whether it offers a service for a given product — an approach that is simply not feasible. Finally, interaction with physical products often happens spontaneously as users are on the go. Users will often discover that a product offers a new service that was previously not known to them. In such a situation, going through the process of installing an application that supports the new service is not desirable. If new software is needed on the mobile phone, it should be deployed on the fly and without disturbing the interaction flow between the user and the physical object.

We believe that the majority of these challenges can be best addressed by a *Browser for the Internet of Things* (BIT). Such a browser, offering a single runtime environment that is home to all sorts of services as described above, addresses two distinct needs (see Figure 4.1):

1. From a user's point of view, BIT represents a one-stop shop for the digital services of a physical product by offering a single point of interaction.

2. From a developer's point of view, BIT provides a software framework that facilitates the creation of product-related digital services.

Several other projects have explored frameworks to facilitate the creation of services that allow users to interact with physical objects

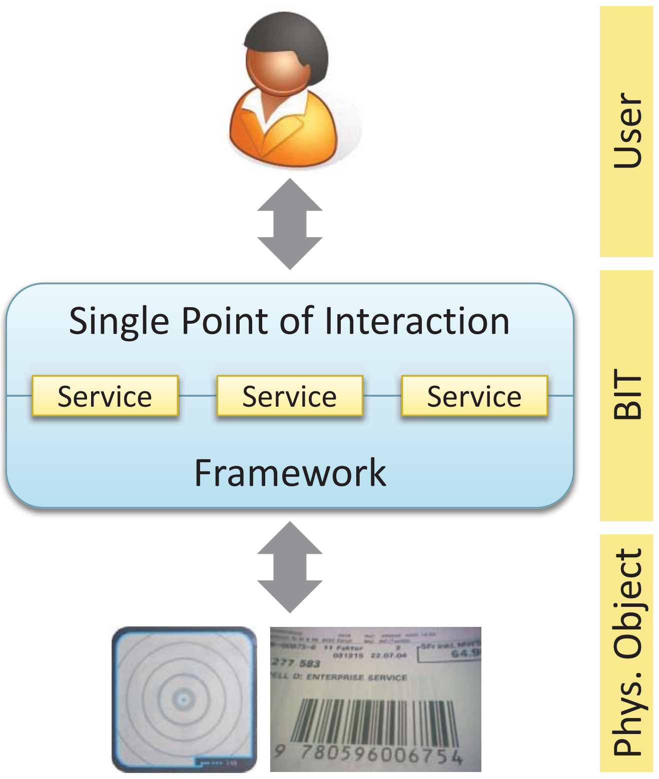


Figure 4.1.: *Browser for the Internet of Things (BIT) overview.* BIT addresses two distinct needs: On the one hand, it represents a "single point of interaction" for users. BIT frees users from the burden of manually executing individual applications and coordinates the many possible services on their behalf. On the other hand, BIT provides a software framework that simplifies the development of services for tagged objects.

through mobile phones. The Physical Mobile Interaction Framework (PMIF) [129] provides developers with a generic framework to write applications that support different interaction techniques, such as touching, pointing, and scanning. It frees developers from the need to deal with the specific technologies used to implement these techniques. The PERCI (PERvasive ServiCe Interaction) project [16] also aims at facilitating physical mobile interaction, however, the focus lies on the automatic generation of user interfaces from service descriptions based on Semantic Web Services. The REACHeS (Remotely Enabling and Controlling Heterogeneous Services) project [120], finally, proposes a system that implements the universal remote control paradigm. NFC tags are used to enable users to physically interact with their environment.

While all of these projects, which will be revisited in Section 4.10, are related to our work, our focus is more on single, low-value physical products than on smart environments. We aim at providing a runtime environment in which services offered by a large number of interested parties can be deployed and executed. In contrast to many other scenarios discussed in the ubiquitous computing community, the physical objects tend to be more inexpensive in our examples, and both tagged products as well as interaction devices are highly mobile. The emphasis of our research is thus on a lightweight approach that allows for the fast and easy creation of new services in this specific domain.

4.3. Scenario

We start our discussion of BIT by defining a concrete scenario of use that forms the basis of our further analysis. The framework that we will present later in this paper will be designed to meet the requirements that can be derived from the use cases in this scenario. For the purpose of this brief description, we consider a character which we name Alex.

As Alex enters his kitchen in the early morning, he realizes that there must have been a power cut last night: Instead of having started up automatically and being ready to brew a coffee, his coffee maker is showing a blinking time indicator only. Alex reaches for his mobile phone and touches the coffee maker. The coffee maker service provided by the manufacturer launches immediately. Alex chooses the "set clock to current time" menu item and touches the device again.

After browsing the newspaper, he heads off to the local grocery store, which has recently introduced a new self-checkout system. He picks up a shopping basket and uses his mobile phone to touch the NFC tag attached to its handle. He proceeds to the dairy products, where he realizes that the store is out of his favorite cheese. There is another one that piques his interest, but since he does not tolerate lactose well, Alex has to watch out when buying cheese. As he scans the package's barcode with his phone's built-in camera, a number of facts, tips, and notes appear on the screen. The topmost item is the familiar green check mark symbol, indicating that the product suits his dietary restrictions. Alex places the cheese in his basket and confirms this action on his phone, which now shows him the running total of the items to be purchased.

In the next aisle, there is a promotional display advertising a new razor. Alex is sceptical, but the introductory price seems attractive. He scans the package's barcode and is surprised to find a rave review by one of his friends who is on the same social networking site like

Alex. But he is also surprised to see that the "introductory price" is not all that low. Another store that is just 200 meters from his current location offers the same product for 20% less, and directions for getting there are shown on a map.

Alex adds a few more products to his basket, always scanning them to keep the running total up to date. Once he is done, he proceeds to the self-checkout station. He completes the transaction by touching the "pay" symbol with his mobile phone, which automatically charges his credit card with the amount due.

Before going back home, Alex decides to visit the public library to find a good book for the weekend. He is an avid reader and a member of three online reading communities. As he browses the collection, he finds an interesting-looking book and wonders what others in his communities think about it. His phone is set up to show their discussions as soon as the corresponding book is scanned. However, he realizes that the "shopping" profile is still active and causes price comparison information to appear as well. Because he does not want to be disturbed by such content, he switches his phone to the "library" profile, which he set up a few months ago to only include the three reading communities.

A little later, he has checked out a new book and is now on his way home. In the bus, he sees a poster advertising a new movie. He is curious and scans the barcode in the corner of the poster. A trailer starts playing on his phone. But Alex decides not to watch it now. To save it for later, he selects the "bookmark" option on his phone, which adds the movie to his collection of remembered products.

Once he arrives back home, he realizes that the trip has made him hungry. He takes a pizza out of his freezer and puts it in the oven. To set the oven to the right temperature and time, he touches it with his mobile phone, selects "read preparation instructions", and scans the barcode on the pizza box. While waiting for the pizza, he sits down at his computer to check his mail. There is a message from a friend recommending a new gadget for his phone. He opens the site in his computer's web browser and finds that it is actually a campaign to encourage people to reduce their carbon emissions. The site says that it offers a "carbon footprint calculator" for the mobile phone, which lets you see how much carbon dioxide was created during an item's production process. And by adding these numbers up, it also lets you keep track of your carbon personal footprint. Alex is intrigued and decides to have a go. The web site now displays a barcode and asks him to scan it with his mobile phone. He does so and selects "shopping" when prompted to select the profile to which he wants to add the carbon footprint calculator.

4.4. Requirements

Based on the above considerations, we can derive a number of requirements that BIT needs to fulfill in order to support the wide array of services that can be implemented for tagged products.

Navigation of services In order to represent a single point of interaction as outlined above, BIT must allow users to navigate the different services that are available. For example, in a retail store context, users often interact with several services at the same time. In this situation, users deal mainly with the store's self-checkout service. Intermittently, however, their interest shifts to other services, such as a third-party price comparison or product review service. Such changes of focus can occur frequently and must be supported accordingly.

Responsiveness As we expect the switching from one service to another to occur frequently, the user interface should be responsive and latency should be minimized.

Coordination of services As we saw before, it is not unusual that a single barcode scan can trigger the activation of multiple services. BIT must coordinate the execution and presentation of the many services that a user deems interesting. In particular, it must ensure that tag reads are made available to all relevant services. By the same token, it must provide mechanisms that prevent services from accessing a shared resource (e.g., the RFID reader), at the same time.

Use without physical object BIT must allow for the use of services even when no tagged object is present. An instance where this is neccessary concerns the configuration of a service. In the scenario outlined above, it should be possible for users to adjust their dietary restrictions without scanning a product. Another example is the movie poster mentioned in the scenario. In some situations, users may scan an object and discover a service they do not want to access instantly. BIT should allow them to remember this context in order to allow for later service access without the physical object being present.

Service invocation types The above scenario describes two different types of service invocation: *direct* and *indirect* invocation. In the example of an in-store service that is offered by a retailer, the service should be presented to the user as soon as the tag attached to the shopping basket is scanned. Its invocation is the direct result of the physical interaction. Different examples are the price comparison or review services. When a user scans a tag, he or she is notified that these services have information available concerning the product. A subsequent service invocation takes place only if the user expresses an interest by manually selecting the service. In other words, the invocation is not the direct result of the physical interaction. BIT must provide mechanisms to support both invocation types.

Discovery Users must be able to discover newly available services by physically interacting with objects. An example from our scenario is the carbon footprint calculator that was advertised through a barcode symbol on the computer screen.

Appliance status recognition When users interact with appliances, it is not enough to simply identify the object. BIT must also be able to retrieve status information from the appliance, such as its current settings or an error code.

Appliance control In addition to recognizing the status of an appliance, BIT must also be able to control the device to, for example, adjust its settings. In the scenario above, this is necessary to update the coffee maker's clock settings. Appliance control requires bidirectional communication between BIT and the appliance and can be implemented using different technologies, such as Bluetooth or NFC. BIT must make these different communication technologies available in an easy-to-use way and free developers from the need to deal with low-level connection handling.

Acquisition of context information Many services depend on context information to adapt their behavior. This is not limited to the recognition of an appliance's status, but also includes the user's location and, in a somewhat relaxed use of the term, users' language and phone settings. Access to such context information from different sources must be made available in a uniform way.

Integration with backend infrastructure In most cases, product-related information and services that are presented by BIT are stored in a backend infrastructure. The open lookup infrastructure discussed in the previous chapter represents such a backend infrastructure. In order to discover and retrieve product-related data, BIT must seamlessly integrate with the open lookup infrastructure. Developers must be freed from the need to manually contact resource repositories, fetch resource descriptions, and parse the replies obtained from repositories. In order to make these steps transparent for developers, BIT must wrap them in methods that are easily and locally accessible by service developers.

Persistent storage Certain services need to store data that must be available during subsequent invocations of the same service. In the above scenario, the allergy checking service must be able to persistently store the user's dietary restrictions, and the carbon footprint calculator needs to keep track of the user's accumulated emissions. On top of this, BIT must also be able to persistently store browser-wide data, such as user settings.

Tagging technology-agnostic A service is usually indifferent to the precise reading capabilities that are available on a device and to the reader that originally generated a read event. Instead, it should be possible for developers to simply process reads of physical objects, irrespective of the tagging technology used.

Platform and device independence As services should run on any platform for which a BIT implementation is provided, device and platform idiosyncrasies must be abstracted. For example, device-specific APIs for the use of RFID or barcode readers as well as location information should be wrapped. Also, it should be possible to deploy a service without needing to create a separate GUI for every available toolkit, form factor, or screen size of the targeted devices. The same applies to other output modalities, such as phone vibration.

Easy-to-learn and rapid service development Many of the people who could create valuable services for physical objects do not have much background in software development, and even less so on mobile devices with their specific intricacies. The development of services for BIT should thus be easy to learn. Ideally, syntax, semantics, and concepts used in BIT should build on what is already familiar to many developers from other domains. In addition, the lifetime of some services which are more of a promotional nature may be limited. Other services require frequent changes. BIT should therefore support rapid prototyping, short development cycles, and iterative design, which will also be beneficial for the usability of services [80].

Design for low attention BIT is used in situations with a very different context than many other programs, even on mobile phones. In virtually all of these situations, the user's attention is split between the physical product, the mobile device, and an environment that abounds with stimuli (such as announcements in a retail store or a chit-chat with a fellow shopper). The design of BIT should consider these factors and minimize the effort that is needed for interaction.

Unified user experience In order to ease interaction with the plethora of services that may be provided, BIT must enforce some restrictions with regard to user interface design. For example, the steps needed to stop the use of a service should always be the same, no matter how the service is implemented. BIT should also facilitate the implementation of common user interface patterns that are shared by most services, in order to save developers the trouble of repeating themselves when writing code.

Security and privacy As we envision BIT to be a single tool that mediates all interaction between a user and every physical product's digital offerings, it would be easy to track nearly every move of a user. Privacy is thus a major concern, and BIT must support users in revealing as little as possible about themselves if they wish to do so. BIT must also ensure that a service can only access those resources for which it is authorized. For example, a service must be constrained to read and write its own storage area only. Likewise, access to context information must be controlled. Finally, BIT must mitigate potential threats originating from malicious services in order to protect the mobile phone itself.

4.5. Implications

From the requirements listed above, we can derive a number of implications that will guide the design of the framework and the architecture of BIT that we will present in the next sections.

4.5.1. Services provided by mobile devices

There are two main architecture blueprints that could be used for the implementation of BIT. In the first architecture, the user's client device can be seen as a terminal to a service running on a remote server. Such an architecture is consistent with the traditional notion of a web browser that gathers user input, forwards it to a remote server, and renders the response received. All application logic is located in the backend. In the second architecture, the client device is not just a terminal, but also the runtime environment for the application logic. In this latter style, the service is entirely provided by the client device and does not rely on any backend infrastructure.

Many systems today mix elements from both architectures and do not clearly fall into one of the two categories. However, the use of Ajax (asynchronous JavaScript and XML) techniques has shown a trend to move the parts of an application that are concerned with user interaction closer to the client device. This trend has been motivated to a great extent by the improved user experience that can be achieved due to better responsiveness. By executing a significant part of the application on the client device, both the amount of data transferred as well as the number of roundtrips required can be reduced.

In the previous section, we demanded that services be responsive and easy to navigate. This indicates that the second architectural style, in which services are entirely provided by the mobile device, is more suitable for BIT than the first one. This choice is further reinforced by the many device capabilities, such as barcode readers and GPS, NFC, or vibration modules, that must be accessed by services. While it is certainly possible to make such capabilities available to a service running on a remote host, latency and responsiveness can again suffer, and it becomes less straightforward to control appliances. Finally, providing services directly on the mobile device has the advantage that they can be accessed even when no internet connectivity is available (e.g, in the subway).

4.5.2. Mobile code rather than built-in protocols

In order for BIT to be able to control an appliance, there must be agreement on the syntax and semantics of communication. One approach to create such agreement is to standardize protocols that all participants (i.e., all appliances and mobile devices) must implement. Another approach is to use mobile code [47] in order to enable interoperability between devices that have very limited prior knowledge about each other [30].

The requirement for BIT to be able to control previously unknown appliances mandates the latter approach, i.e., the use of mobile code. It is very hard to imagine that a large number of very different appliances would allow for control through the same standard protocol. Even if such a protocol was available, it would severely limit flexibility and leave little room for innovation. Instead of relying on a few common protocols, BIT leverages mobile code in order to implement the services offered by tagged objects.

As a consequence, the service management mechanisms incorporated in BIT must be designed accordingly. As soon as users indicate that they want to invocate a service, BIT needs to obtain and execute the corresponding code. Since many services will be invoked more than once, their code should be cached on the phone. Finally, BIT must regularly check if updates are available for a given service.

4.5.3. Full programmability

For the implementation of services, BIT uses a combination of both a declarative markup language to describe the user interface as well as a full-fledged scripting language. In an earlier prototype of BIT [18], we explored ways to implement services based on a custom markup language only. However, it became quickly apparent that this approach did not offer the flexibility needed. For example, services such as self-checkout or the carbon footprint calculator outlined in our scenario could not be implemented based on a purely declarative approach.

We chose a scripting language because it is easy to learn and because it ensures a rapid development process with short iteration cycles where changes are visible immediately. We decided to leverage Lua [62, 61], an existing and comprehensive language, rather than introduce our own scripting language in an effort to provide a simpler, stripped-down language that supports a minimum feature set only. As

Myers et al. note [97], previous attempts at “simplified” scripting languages for writing interactive applications have mostly failed due to a lack of full programmability offered by general-purpose control structures, such as variables, loops, and conditionals.

4.5.4. Minimal attention user interface

We stipulated in the previous section that BIT should be designed for low attention. We will take this requirement into account by building a *minimal attention user interface* as proposed by Pascoe et al. [112]. BIT must follow this principle by, for example, allowing users to access frequently used functions without fixing their eyes on the display first and hiding less relevant information whenever possible.

4.6. Framework Core Concepts

The BIT framework is built around a number of concepts. In this section, these concepts will be presented.

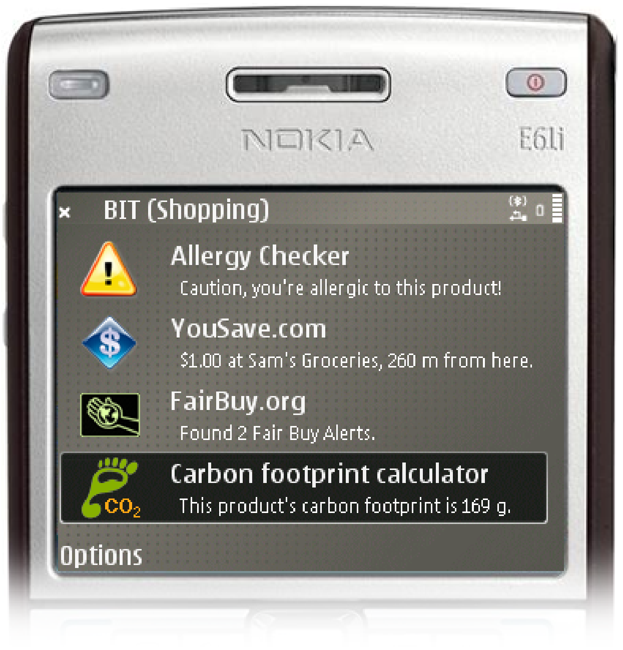
4.6.1. Open lookup infrastructure

BIT leverages the open lookup infrastructure described in the previous chapter. Services offered by the browser will normally retrieve product-related information from one or more resource repositories. Typically, a service provider will maintain its own resource repository, which is used as a data source for the service offered within the browser.

Product manufacturers who wants to fully support services based on BIT should set up their own resource repository and publish a resource description adhering to the “basic” profile for every product. This resource description contains some basic information on a product:

- product name
- manufacturer name
- small product image
- product’s bidirectional communication capabilities (e.g., Bluetooth address), if any

The browser uses these data to display the product’s name and possibly its image as soon as it is recognized through an auto-id tag. Also,



BIT (Shopping)

Allergy Checker
Caution, you're allergic to this product!

YouSave.com
$1.00 at Sam's Groceries, 260 m from here.

FairBuy.org
Found 2 Fair Buy Alerts.

Carbon footprint calculator
This product's carbon footprint is 169 g.

Figure 4.2.: Browser aggregation perspective showing overview of information and services available for a tagged object.

information regarding communication capabilities is used when a service needs to exchange data with an appliance (e.g., to adjust a coffee maker's time settings in the scenario above).

However, the "basic" resource description is not mandatory, and a browser can easily handle objects for which it is not provided. In this case, the only restriction is that the browser itself cannot display the product's name and image or communicate with the object. A particular service may still be able to display such content, which it obtains through some other resource repository.

4.6.2. Perspectives

BIT's user interface is divided into two major parts: the *aggregation perspective* and the *exclusive perspective*. In response to scanning a tagged product, the aggregation perspective provides an overview of information and services that are available. As its name suggests, this perspective aggregates data from various sources in a single view. Figure 4.2 shows an example of BIT's aggregation perspective.

In addition to listing the information and services that are available, the aggregation perspective also displays basic data on the tagged object, such as its name, its manufacturer, as well as a small image of the product (see Section 4.6.1).

As soon as a user selects an item in the aggregation perspective, the corresponding service is opened in the exclusive perspective for a more detailed view. In this perspective, the service can exclusively use the full assigned screen area to display more detailed information (e.g., directions to another store selling the same product for less). Unlike the aggregation perspective, the exclusive perspective also offers the possibility for user interaction and, for example, to collect user input.

A perspective is either active or inactive, and, at all times, there is exactly one perspective which is active. While the fairly small screen estate of mobile phones does not usually permit several perspectives to be shown at the same time, a browser implementation may opt to do so.

4.6.3. Applets

Services and information regarding a physical product are provided through *applets*, i.e., small programs that are executed within BIT. The functionality offered by an applet can range from the very simple task of displaying some information on a physical object to more complex procedures, such as gathering user input and communicating with a physical appliance.

4.6.4. Runlists

Applets can be part of a *runlist*. A runlists groups a number of applets that a user considers relevant in a certain everyday context. For example, while users may be interested in product reviews during a shopping trip, they may prefer not to see such information while at home.

A runlist is an ordered list of applets and is *executed* when a tagged object is detected. Upon execution of the runlist, BIT invokes all applets constituting the runlist, passing the tag read to every applet. The applet is expected to generate a terse output (e.g., “Caution: contains peanuts”) that can be empty optionally. Based on all collected non-empty output returned by the applets, BIT generates a list of results that is organized according to the order defined in the runlist. The results list is then displayed in the aggregation perspective as depicted in Figure 4.2. The example in said figure has been generated by a runlist with an “Allergy Checker” applet placed at its top. By assigning this applet a prominent position in the runlist, the user ensures to immediately grasp the information that is most important to her or him. If the applet does not return any output (e.g., the product does not contain allergy-causing ingredients), it is omitted from the results list.

Users can define any number of runlists with an applet being part of several runlists at the same time. Runlists have a name and are typically defined for situations such as “home”, “shopping”, or “work”. Existing work in the field of activity recognition could be leveraged to automatically select the runlist reflecting a user’s momentary situation. This, however, is beyond the scope of this thesis. Exactly one runlist must be enabled at all times.

4.6.5. Applet dimensions

Applets can be characterized in three dimensions: *mode*, *type*, and *state*.

Mode: aggregation vs. exclusive

Applets can be executed in two different modes: *aggregation mode* or *exclusive mode*. An applet that is invoked in aggregation mode is expected to produce a terse output that can be included in the results list, which is assembled during the execution of a runlist and displayed in the aggregation perspective. This output consists of a title, a short description, as well as an image (see Figure 4.2). If an applet chooses not to return any output, it is omitted from the aggregation perspective.

By contrast, an applet running in exclusive mode has access to the entire exclusive perspective as described in Section 4.6.2. Unlike in aggregation mode, it is not limited to producing output, but can also collect user input.

Type: permanent vs. transitory

While some applets can be used in connection with any tagged object, others can provide a meaningful service only in connection with a specific one. An example of the first kind is an applet that offers product reviews. It can show reviews for a potentially large number of products and should be invoked whenever the user scans a tagged object. This is ensured by including the applet in a runlist, which requires it to be downloaded and installed in the browser for frequent use. In BIT, such applets are called *permanent* because of their long lifetime in the browser. In fact, permanent applets remain installed until they are manually uninstalled by the user.

An example of the second kind is an applet to control an appliance. This sort of applet can provide a service only for a specific tagged object, such as a particular coffee maker. Including it in a runlist makes little sense, since it cannot react in a meaningful way in response to the recognition of any other object. On the other hand, such an applet should start automatically as soon as the user interacts with the tagged object, even if the appliance is encountered for the first time. In order to permit such spontaneous interaction, BIT must download and execute the applet immediately without requiring its installation first. Because it is not installed, the applet is disposed of as soon as interaction ends, i.e., when it is closed by the user. As the lifetime of such applets within the browser is very limited, they are called *transitory*.

Permanent applets do not have an association with a particular tagged object,[1] since they can be run following the scan of an arbitrary object. For transitory applets, however, there is an association with a particular tagged object. This association can be seen as privileged, as it causes the browser to automatically launch an applet in response to a scan. Because of this privileged access to the browser, only a product's manufacturer can provide a transitory applet. This avoids the risk of unsolicited applets popping up as soon as an object is scanned.[2]

Unlike permanent applets, transitory applets cannot persistently store data in order to have them available during a subsequent execution. After a transitory applet is stopped, it is removed entirely from the browser without leaving any traces behind. This restriction represents a safeguard that enables spontaneous, possibly one-off interaction with services whose source cannot necessarily be trusted.

In summary, permanent applets must be installed in the browser at the user's explicit request prior to their use. They remain in the browser until they are uninstalled. All applets that are part of a runlist are of the permanent type. A transitory applet, by contrast, will not be installed in the browser. It is downloaded and started automatically by BIT in a direct response to the user's interaction with a tagged object. Transitory applets cannot be part of a runlist and, as a result of this, can only be executed in exclusive mode.

The main differences between the two types are summed up in Table 4.1.

[1] Tied applets, which are discussed below, represent an exception from this rule.

[2] This does not prevent arbitrary third parties from providing information on a tagged object, which is still possible through permanent applets as well as the search service discussed in the previous chapter. The integration of search services in BIT will be further detailed below.

	Permanent applets	**Transitory applets**
Installation	necessary	not necessary
Inclusion in runlist	possible	not possible
Execution in aggregation mode	possible	not possible
Execution in exclusive mode	possible	possible
Coupled state	possible	possible
Uncoupled state	possible	not possible
Tying to tagged object	possible	not possible
Storage read access	allowed	not allowed
Storage write access	allowed	not allowed
Can be provided by	any party	manufacturer
Open lookup infr. profile name	"permanent-applet"	"transitory-applet"

Table 4.1.: Differences between permanent and transitory applets.

State: coupled vs. uncoupled

As we saw before, the results shown in the aggregation perspective are the result of a user-initiated scan. If one of the items is selected, the corresponding applet is executed in exclusive mode. In this case, the applet is said to run in *coupled* state because it is logically "coupled" with the tagged object that was recognized earlier.

However, permanent applets can also be started manually by the user, i.e., without scanning an object first. This is the case, for example, when a user wants to adjust the dietary restrictions in an "allergy checker" applet. In this situation, the applet runs in exclusive mode and, since the execution is not the result of an object being recognized, in *uncoupled* state.

The relations between these three dimensions of an applet — mode, type, and state — are visualized in Figure 4.3. Note that the type dimension is a static property of an applet, while the other two dimensions can change during its lifetime.

4.6.6. Tying applets to tagged objects

BIT allows users to "tie" a permanent applet to a specific tagged object. If an applet is tied to a tagged object, it will be started automatically in exclusive mode whenever the object is recognized by BIT. The effect is the same as if the user had manually selected the applet in the

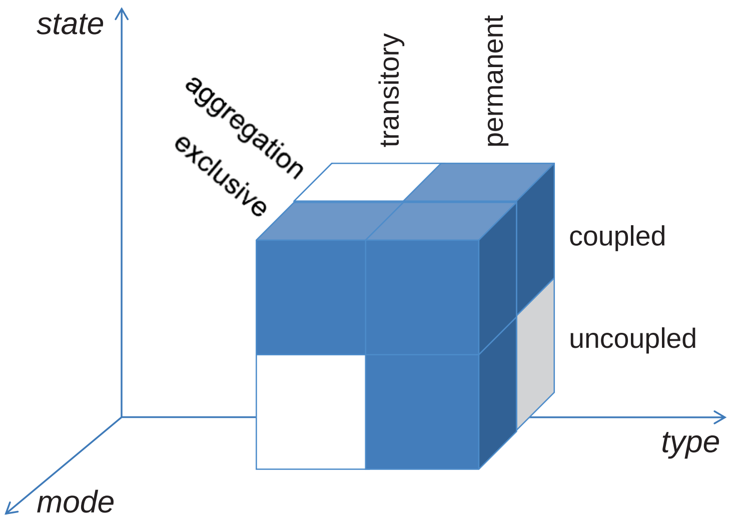


Figure 4.3.: Relations between the three dimensions of an applet. Only the combinations that are visibly highlighted can occur.

aggregation perspective after the object was scanned.

This feature is somewhat similar to the automatic startup of transitory applets. However, the two differ in that the option of having applets launched automatically by making it transitory is only available to the manufacturer of a tagged object. In addition, a tied applet has unrestricted access to storage provided by the browser, which is not the case for transitory applets.

While an applet can be tied to any number of tagged objects, a tagged object can have at most one applet tied to itself.

4.6.7. Virtual reads through bookmarks and history

Similar to a web browser, BIT allows users to bookmark tagged objects. It also keeps a history of all tagged objects that were scanned. With these features, users can access the services provided by a tagged object without actually having the physical product at hand. When a bookmark or history item is selected, BIT internally creates a virtual read, which is processed as if the physical object had been recognized. For an applet, a virtual read is thus indistinguishable from a physical read.

4.6.8. Handling of tagged object reads

As we saw in Section 4.6.4, the browser handles the recognition of a tagged object by executing the currently active runlist and passing the tag read to all applets contained in the runlist. We further saw in Section 4.6.5 that transitory applets, which cannot be part of a runlist, are started immediately as a tagged object is recognized.

In this section, we will present in detail how BIT proceeds when the user scans a tagged object. As we will see, BIT is tightly integrated with the open lookup infrastructure described in Chapter 3 and leverages resource repositories to locate applets during the processing of reads. The exact steps taken by the browser in this phase depend on whether the tagged object was recognized while the aggregation perspective or the exclusive perspective was active.

Aggregation perspective

When a tagged object is recognized while the aggregation perspective was active, BIT starts two threads (see Figure 4.4). In the first thread, the runlist is executed and the aggregation perspective is updated as described in Section 4.6.4. In the other thread, the browser first checks whether there is an applet that is tied to the tagged object that was recognized. If this is the case, BIT starts the applet in exclusive mode and switches from aggregation perspective to exclusive perspective.

If there is no tied applet, BIT checks whether the manufacturer of the recognized tagged object provides an applet for it. Since applets themselves are considered resources in the sense of the open lookup infrastructure, BIT must contact the product manufacturer's resource repository for this check. The resource repository is located through the open lookup infrastructure's manufacturer resolver service as described in Section 3.4.3. Note that in this context, the term "manufacturer" has a somewhat broader meaning. It simply refers to the party that commissioned (i.e., reserved) an auto-id identifier for the physical object. In the case of a movie poster, for example, the "manufacturer" would typically be an advertising company, a movie distributor, or a similar entity. Once the repository is located, BIT submits a query consisting of the tag id that was read as well as the profile names "permanent-applet" and "transitory-applet". If a resource description is returned, indicating that an applet is available, the browser proceeds as follows:

- *For transitory applets:* The applet is downloaded and executed immediately in exclusive mode.
- *For permanent applets:* The applet is downloaded, installed, and

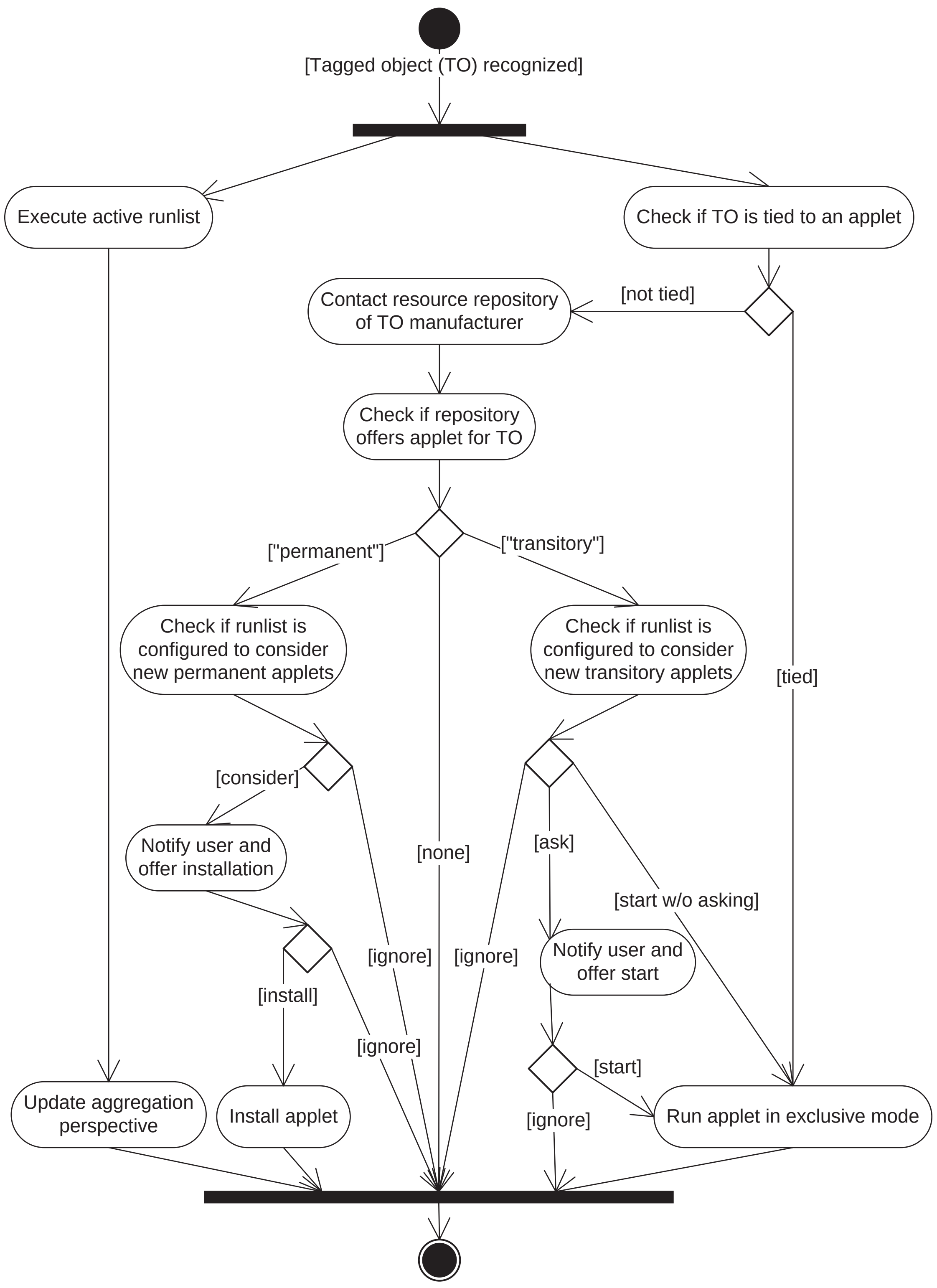

[Tagged object (TO) recognized]
Execute active runlist
Check if TO is tied to an applet
[not tied]
Contact resource repository of TO manufacturer
Check if repository offers applet for TO
["permanent"]
["transitory"]
Check if runlist is configured to consider new permanent applets
Check if runlist is configured to consider new transitory applets
[tied]
[consider]
Notify user and offer installation
[none]
[ask]
[start w/o asking]
[ignore]
[ignore]
Notify user and offer start
[install]
[ignore]
[start]
[ignore]
Update aggregation perspective
Install applet
Run applet in exclusive mode

Figure 4.4.: UML activity diagram showing the steps taken by the browser when a tagged object is recognized while the aggregation perspective is active.

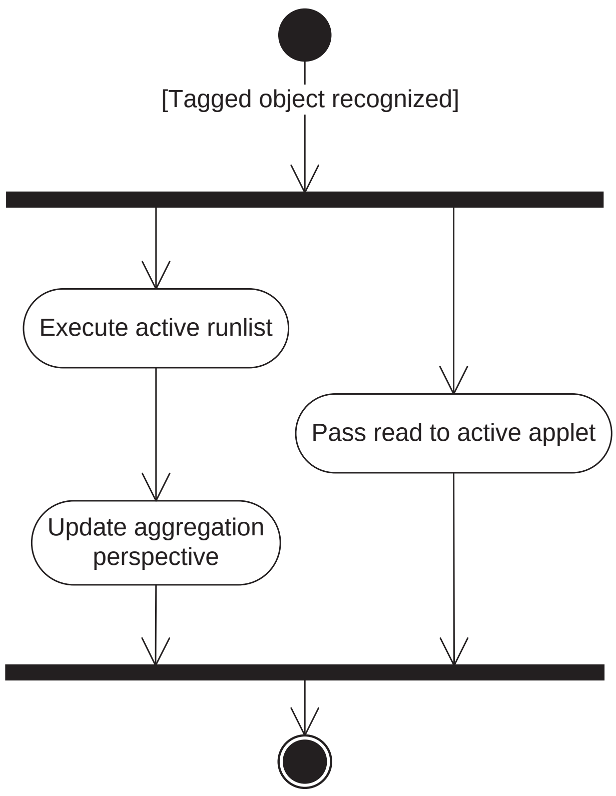

[Tagged object recognized]
Execute active runlist
Pass read to active applet
Update aggregation perspective

Figure 4.5.: UML activity diagram showing the steps taken by the browser when a tagged object is recognized while the exclusive perspective is active.

becomes part of one or more runlists, depending on the user's choice. It is not executed at that time.

Whether download and installation / execution actually take place depends on the configuration of the runlist and possibly the user's choice. For convenience, runlists can be configured to always proceed with the installation / execution, to prompt the user first, or to entirely ignore permanent and transitory applets that are associated with a tag.

Exclusive perspective

Like in the aggregation perspective, BIT starts two threads when a tagged object is recognized while the exclusive perspective is active (see Figure 4.5). Again, the first thread is concerned with executing the runlist and updating the aggregation perspective with the resulting items.[3] The other thread, however, simply passes the tag read to the applet running in the exclusive perspective.

[3]The aggregation perspective is updated to ensure that the output produced by other applets is available immediatly when the user switches from exclusive to aggregation perspective.

4.6.9. Endpoints

BIT provides applet developers with the abstraction of communication *endpoints*. An endpoint represents a logical connection to a physical object that offers bidirectional communication through Bluetooth, TCP/IP, HTTP, NFC, or a similar technology. It allows developers to write applets that can communicate with physical objects without needing to worry about the concrete underlying communication technologies.

When a physical object is recognized, the browser detects whether the object has bidirectional communication capabilities or not. It does so by inspecting the "basic" resource decription, which is available through the object manufacturer's resource repository (see Section 4.6.1). If the object was originally recognized via the phone's NFC reader, the browser also initiates the NFC connection handover protocol [103] to probe for additional communication capabilities.

If the object is found to offer bidirectional communication, an endpoint is created and passed to the applet. As soon as the applet decides to send or receive a message to or from the endpoint, the browser automatically establishes a physical connection to the device through one of the communication technologies that were discovered when the physical object was originally recognized.

4.7. Service Development

After having reviewed the core concepts of the BIT framework, in this section, we will present the tools that BIT provides for developers to implement services. The major building blocks are:

- the BIT markup language (BITML) for the definition of services' user interfaces;
- a scripting language for the implementation of the dynamic behavior of services;
- the BIT API, which allows services to access the functionality provided by the browser.

4.7.1. BIT markup language (BITML)

As we saw in Section 4.4, BIT services must be available on different mobile platforms, which usually include a proprietary GUI toolkit. To ensure the portability of services across platforms, BIT relies on its own user interface description language named *BITML* (BIT markup language). BITML is based on XML and defines a number of GUI widgets which are commonly used in user interface design (e.g., text, images, input fields, lists, etc.). The presentation of these widgets can be customized using Cascading Style Sheet (CSS) [12] formatting. When BITML is processed by the browser, it is rendered using the platform's native GUI controls.

Researchers have proposed a plethora of user interface markup languages, with the User Interface Markup Language (UIML) [2] and the Extensible Interface Markup Language (XIML) [115] perhaps being the best-known examples. Mozilla's XML User Interface Language (XUL) [52] and Microsoft's Extensible Application Markup Language (XAML) [90] are widely used examples from the industry. BITML shares many similarities with such languages, even though it is much simpler, but also less powerful. BITML does not aim to reinvent the wheel and is part of BIT because of its simplicity. In fact, it could be easily replaced with another language, such as XUL. However, as the focus of this thesis does not lie on portable user interfaces, we did not implement such an extensive standard.

4.7.2. User interface organization and GUI widgets

In BITML, an applet's user interface is structured into *views*. Each view occupies the entire screen estate that is available to the browser's exclusive perspective, which, in our implementation (see below), is identical with the device's full screen. Views can have a *title*, a *body*, and a *menu*. The body of a view must be either a *list*, a *form*, or *media content*, i.e., text and images. Figure 4.7.2 shows the source code of an example view along with the resulting user interface. The view shows the items in the shopping cart of a self-checkout applet. It has a list as its body and contains two menus. One menu relates to the view itself, while the other menu relates to the list. In other words, the second menu represents a context menu for the list items.

BITML offers a number of GUI widgets through the following XML elements:

- `<view>`

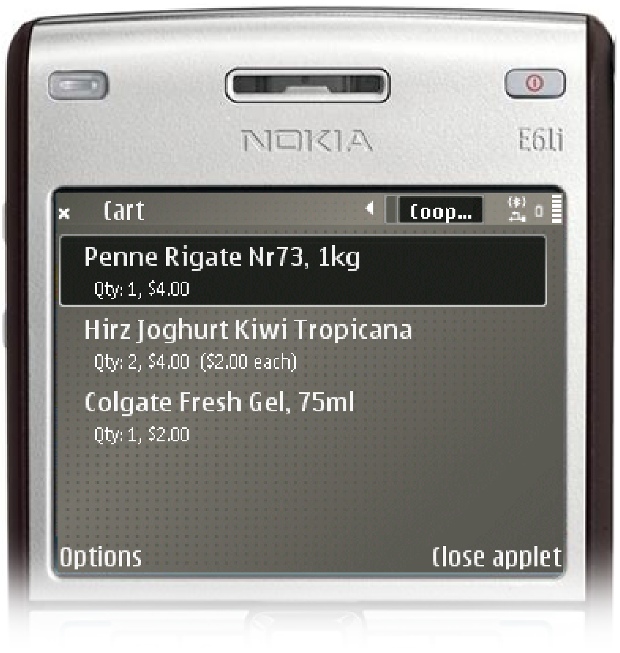
  This element represents a view, i.e., the main container for GUI

```
<view title="Cart">

  <list>
    <list_item description="Qty: 1, $4.00">Penne Rigate Nr73, 1kg</list_item>
    <list_item description="Qty: 2, $4.00 ($2.00 each)">Hirz Joghurt Kiwi
        Tropicana</list_item>
    <list_item description="Qty: 1, $2.00">Colgate Fresh Gel, 75ml</list_item>
    <menu>
      <menu_item action="change_quantity(index)">Change quantity</menu_item>
      <menu_item action="delete(index)">Remove from cart</menu_item>
    </menu>
  </list>

  <menu>
    <menu_item action="bit.show_view('total', {})">Show cart total</menu_item>
    <menu_item action="checkout()">Checkout</menu_item>
  </menu>

</view>
```



Cart

Penne Rigate Nr73, 1kg
Qty: 1, $4.00

Hirz Joghurt Kiwi Tropicana
Qty: 2, $4.00 ($2.00 each)

Colgate Fresh Gel, 75ml
Qty: 1, $2.00

Figure 4.6.: *Example view.* Source code and rendered output.

widgets. All other BITML elements must be — directly or indirectly — enclosed in a view.

Can contain: `<menu>`, `<list>`, `<form>`, `<media>`

Is child of: —

Attributes:

– `title`: An optional title for the view.

– `action`: Scripting code to be executed when the view is no longer needed (e.g., user has aborted or form has been committed).

- `<menu>`

  Adds a menu to a view or a list. No more than one menu is allowed per view or list.

  Can contain: `<menu_item>`, `<submenu>`

  Is child of: `<view>` or `<list>`

  Attributes: —

- `<menu_item>`

  Adds a menu item to a menu.

  Can contain: —

  Is child of: `<menu>`

  Attributes:

  – `action`: Scripting code to be executed then the menu item is selected.

- `<submenu>`

  Adds a submenu to a menu.

  Can contain: `<submenu_item>`

  Is child of: `<menu>`

  Attributes: —

- `<submenu_item>`

  Adds a menu item to a submenu.

  Can contain: —

  Is child of: `<submenu>`

  Attributes:

  – `action`: Scripting code to be executed then the menu item is selected.

- `<list>`

  Fills the view with a list.

  Can contain: `<list_item>`

Is child of: `<view>`

Attributes: —

- `<list_item>`

  Adds an item to a list.

  Can contain: —

  Is child of: `<list_item>`

  Attributes:

  – `description`: Optional description for a list item, which will be displayed by the browser.

  – `selected`: Can be set to “true” to preselect a list item. Default value is “false”. No more than one item can be set to “true” in a list.

- `<form>`

  Fills the view with a form.

  Can contain: `<form_item>`

  Is child of: `<view>`

  Attributes:

  – `action`: Scripting code to be executed when the form is committed.

- `<form_item>`

  Adds an item to a form. An item has a value that can be entered / modified by the user. An initial value can be provided as the element’s content.

  Can contain: `<radio_item`, `<checkbox_item>`

  Is child of: `<form>`

  Attributes:

  – `type`: Defines the type of the item and can be “text” for a text field, “integer” for an integer field, “float” for a float field, “date” for a date field, “time” for a time field, “radio” for a group of radio buttons, and “checkbox” for a group of check boxes.

  - `label`: Can be set to a string that is used as a caption for the form item.

- `<radio_item>`

  Adds a radio button to a radio button group (i.e., a form item).

  Can contain: —

  Is child of: `<form_item>`

  Attributes:

  - `selected`: Can be set to "true" to preselect the radio button. The default value is "false".

- `<checkbox_item>`

  Adds a check box to a check box group (i.e., a form item).

  Can contain: —

  Is child of: `<form_item>`

  Attributes:

  - `selected`: Can be set to "true" to preselect the radio button. The default value is "false".

- `<media>`

  Fills the view with media content, i.e., a mix of text and images.

  Can contain: `<text>`, `<image>`

  Is child of: `<view>`

  Attributes: —

- `<text>`

  Renders a text section.

  Can contain: `<break>`

  Is child of: `<media>`

  Attributes: —

- `<break>`

  Creates a line break in a text section.

  Can contain: —

  Is child of: `<text>`

  Attributes: —

- `<image>`

  Renders an image.

  Can contain: —

  Is child of: `<media>`

  Attributes:

  – `source`: URL or local file name of image to be rendered. The PNG, JPEG, and GIF formats are supported.

4.7.3. Presentation

BITML itself allows developers to describe the content and structure of their user interfaces. However, it does not provide any means to define their visual presentation. Like many other systems (including HTML, XUL, and UIML), BITML relies on *stylesheets* to specify the presentation characteristics of the user interface. BITML supports Cascading Style Sheets (CSS) [12] as defined by the W3C. The following listing illustrates the use of CSS and BITML:

```
<view>
  <media>
    <text style="text-align: center; font-weight: bold">
      This text will be bold and centered.
    </text>
  </media>
</view>
```

4.7.4. Scripting

For the implementation of the dynamic behavior of services, BIT leverages the Lua scripting language [61, 62]. The browser incorporates a complete Lua interpreter, which supports the local execution of applets. With respect to the requirements set out in Section 4.4, this has several advantages:

- Services can be executed locally and can handle user actions without communicating with a remote server.
- The Lua interpreter represents an additional layer between the service and the underlying mobile phone platform, which can be leveraged to implement security controls.

- As the browser ships with its own built-in interpreter, applets do not rely on a specific programming language provided by the underlying mobile phone platform and are thus platform-independent.

The browser uses Lua for a number of reasons. As a lightweight language, it is well suited to run on mobile devices with limited resources. Moreover, Lua was explicitly designed to be embedded in other applications, such as our browser. Finally, Lua is available in ANSI C, which makes it readily portable to various mobile phone platforms. Lua is free and open source software.

In the next few paragraphs, we will give a very brief overview of the Lua language in order to provide the background needed to follow the examples discussed below.

Lua is a dynamically typed language. The main types provided are `string`, `number` (accommodating both integer and floating point values), `boolean`, and `table`. Tables play a major role in Lua. Tables implement associative arrays and can be used to store key/value pairs, values only (i.e., act as an array), or a mix of the two. The following example illustrates this:[4]

```
-- Use table as an array:
fruits = {"apple", "pear", "orange"}
print(fruits[1]) -- output is "apple"

-- Use table as an associative array:
tastes = {strawberry = "sweet", lemon = "sour"}
print(tastes["lemon"]) -- output is "sour"

-- Syntactic sugar:
print(tastes.lemon) -- output is "sour"
```

Variables in Lua are generally global unless they are explicitly declared as local. Before the first assignment to a variable, its value is `nil`. Lua's operators include `==` to test for equality, `~=` to test for inequality, and `..` to concatenate two strings. Lua also uses `#` as a unary operator which returns the length of a string or a table (e.g., `#fruits` in the above example would return 3). Indexes in Lua are based on 1, i.e., the first element of array `a` can be found at `a[1]`.

4.7.5. Scripting in views

Like in many web development frameworks (e.g., Ruby on Rails, PHP, JSP), markup and scripting code can be mixed to create a dynamic

[4] A double hypen (`--`) is used in Lua to introduce a comment.

user interface. BIT uses XML processing instructions to wrap scripting code that can be inserted anywhere in the markup code. The following listing provides an example:

```
<view>
  <list>
  <?
    items = {"apple", "pear", "orange"}
    for i = 1, #items do
      bit.out("<list_item>" .. items[i] .. "</list_item>")
    end
  ?>
  </list>
</view>
```

Before this BITML code is rendered, the scripting fragment in the processing instruction (`<? ... ?>`) is executed. The script uses the `bit.out` function to collect its output. BIT replaces the processing instruction in the original BITML code by this output and renders the resulting code in a second step. The above example results in the following BITML code to be rendered:

```
<view>
  <list>
    <list_item>apple</list_item>
    <list_item>pear</list_item>
    <list_item>orange</list_item>
  </list>
</view>
```

Additionally, scripting code is expected by the `action` attribute of some BITML elements, such as `<menu_item>`. The following example illustrates its use:

```
<view>
  <menu action="bit.show_note(items[index], 'info')">
  <?
    items = {"apple", "pear", "orange"}
    for i = 1, #items do
      bit.out("<menu_item>" .. items[i] .. "</menu_item>")
    end
  ?>
  </menu>
</view>
```

When a menu item is selected in this example, the Lua code in the `action` attribute is executed, which shows the menu item's name (i.e., “apple”, “pear”, or “orange”) on the screen.[5] Note the use of the special

[5]The `bit.show_note` function is part of the BIT API, which is discussed in Appendix B. When it is called, it causes the browser to show a simple message (passed in as the first argument) on the screen. The second argument specifies the type of message and is “informational” in the example here.

variable `index`. Before the script in the `action` attribute is run, the `index` variable is set to the numerical index of the selected menu item.

4.7.6. Applet organization

As we have seen above, views in BIT are described using BITML, and Lua scripting can be embedded into views. We will now review all constituents of an applet and describe how they are bundled into a single entity that can be deployed into the browser.

Technically, an applet is a ZIP file that follows a fixed structure. An applet's ZIP file must contain the following three directories:

- **"views".** This directory contains all views of the applet. For every view, a separate file must be created. File names must end in ".bit", with the part before the ".bit" suffix defining the view's name.
- **"scripts".** This directory can contain Lua scripts. These scripts must contain pure Lua code and can be executed by BIT before any view is rendered. The purpose of these scripts is to allow developers to set up functions and variables that can be used by the Lua scripting embedded in views. While it is possible to define function declarations directly in a view (by embedding them just like any other Lua code as shown above), placing such code in separate files is preferable in order to keep views lean. If this directory contains a file named "main.lua", it will automatically be executed before the processing of any views. Other scripts that are placed in this directory are not executed automatically, but can be manually included from within the main script.
- **"resources".** This directory can contain any additional files that the applet may need, such as images that can be referenced by a view.

The following listing shows the structure of a basic applet consisting of two views named "overview" and "details" as well as a single Lua script to set up functions and variables:

```
views/:
   overview.bit
   details.bit
scripts/:
   main.lua
resources/:
```

4.7.7. Callback functions and applet startup

During an applet's lifetime, the browser invokes callback functions to allow it to react to a number of events. BIT offers five callback functions which can be implemented by applet developers:

- `run_aggregation_mode(tagged_object)`

  This function is used to notify the applet that the object represented in `tagged_object` was recognized (a detailed description of this object is given in Section 4.7.8). By calling this function, the browser also asks the applet to produce some terse output for inclusion in the result list that is shown in the aggregation perspective.[6]

- `start_exclusive_mode_coupled(tagged_object)`

  This function is called when the applet is to be invoked in exclusive mode after the detection of a tagged object (hence the reference to "coupled" in the function name). An implementation of this function typically consists of two steps: First, it prepares some information based on `tagged_object`, i.e., the recognized object. Second, it calls the `bit.show_view(view_name)` function, which asks the browser to show the view indicated by `view_name`. The view will then display the information prepared before. In summary, this function can be likened to a controller in the model-view-controller (MVC) pattern.

- `start_exclusive_mode_uncoupled`

  This function is called when the applet is to be invoked in exclusive mode without a tagged object having been recognized. A typical implementation of this function calls `bit.show_view(view_name)` to display a view that allows users to configure the applet.

- `finish_exclusive_mode`

  The browser calls this function to notify the applet that the user has requested for it to be closed. Typically, this function's implementation performs clean-up tasks, such as storing data persistently.

- `process_read(tagged_object)`

[6] The applet can return its output by calling the `set_aggregation_text` and `set_aggregation_image` functions described in Section 4.7.9.

Function	**Perm. applets**	**Trans. applets**
`run_aggregation_mode`	mandatory	n/a
`start_exclusive_mode_coupled`	optional	mandatory
`start_exclusive_mode_uncoupled`	optional	n/a
`finish_exclusive_mode`	optional	optional
`process_read`	optional	optional

Table 4.2.: This table shows the callback functions that can or must be implemented by the two applet types. Two functions are only available to permanent applets.

This function is invoked to notify an applet that is already running in exclusive mode about the recognition of a new tagged object. The applet is free to react to such notifications in any way or to simply discard them.

A typical applet implements at least the `run_aggregation_mode( tagged_object)` and `start_exclusive_mode_coupled(tagged_ object)` methods by placing them into the applet's "main.lua" file.

While some callback functions can be implemented in both transitory and permanent applets, others can only be provided for either type. The differences between the two are summed up in Table 4.7.7.

4.7.8. The "tagged object" argument

As we saw in Section 4.7.7, some callback functions accept an argument named `tagged_object`. This argument is used by the browser to pass in a Lua table containing information on the recognized object that caused the invocation of the function. The table has the following fields:

- **Tag ID.** This field represents the tagged object's identifier, such as its GTIN number or its EPC. It corresponds with the identically named field from Section 3.4.1.
- **Context.** This field is itself a Lua table that contains all context elements that were supplied by the physical object upon recognition. An example for such a context element is the status code of an appliance that is transmitted over NFC. This field again corresponds with the identically named field from Section 3.4.1.

- **Endpoint.** This field represents an endpoint object that can be used to communicate with the physical object as described in Section 4.6.9.

4.7.9. BIT API

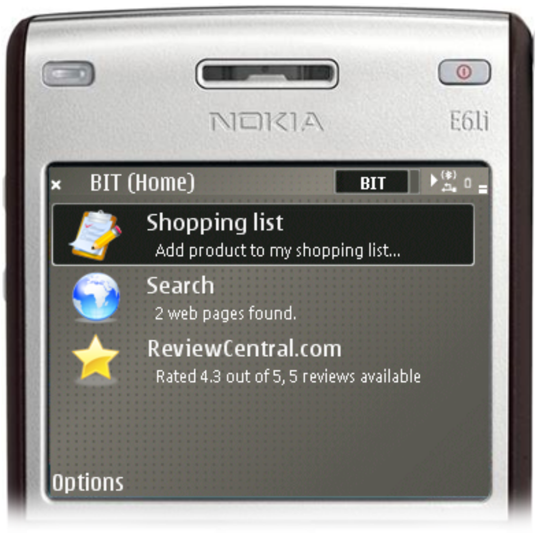
In addition to BITML for the creation of user interfaces and Lua scripting for the implementation of an applet's dynamic behavior, BIT also offers an API. This API provides a range of functionality including the following areas:

- **User interface.** As briefly discussed above, the API provides functions to switch between views and set the output of an applet running in aggregation mode. On top of this, it also contains functions to show basic dialogs that can optionally prompt users for simple input. These functions can be used instead of a view if only basic information needs to be communicated or collected (e.g., to show an error message). Finally, the API offers functionality to show a URL in a web browser and to access the phone's vibration module, if available.
- **Persistent storage.** The API provides functions for an applet to persistently store data.
- **Data exchange.** In order to facilitate data exchange with various backend services, the API offers functionality to invoke HTTP requests and to serialize and deserialize Lua tables to and from the XML and JSON [24] formats. This frees applets from implementing their own parsers.
- **Open lookup infrastructure.** The API contains a number of functions that allow applets to find resources through the open lookup infrastructure. Its entire functionality is available through the API, which wraps the open lookup infrastructure's protocol.

A complete description of the BIT API including a list of all functions can be found in Appendix B.

4.7.10. Example applet

We conclude this section by presenting the example of a product review service. This practical example along with its full source code will illustrate how the different aspects discussed above fit together in applet development.


BIT (Home) BIT

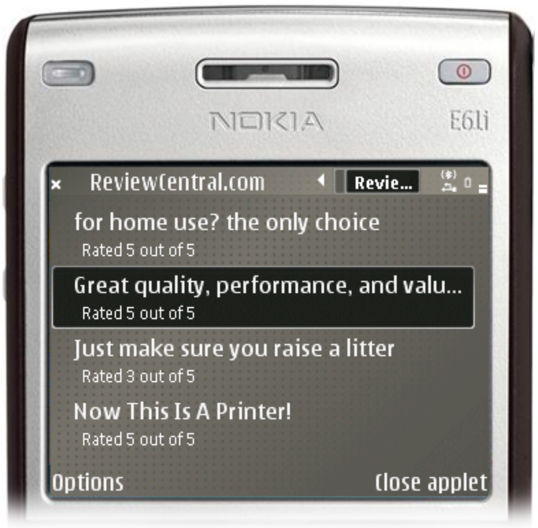
Shopping list
Add product to my shopping list...

Search
2 web pages found.

ReviewCentral.com
Rated 4.3 out of 5, 5 reviews available

Options

(a) Output in aggregation perspective.


ReviewCentral.com Revie...

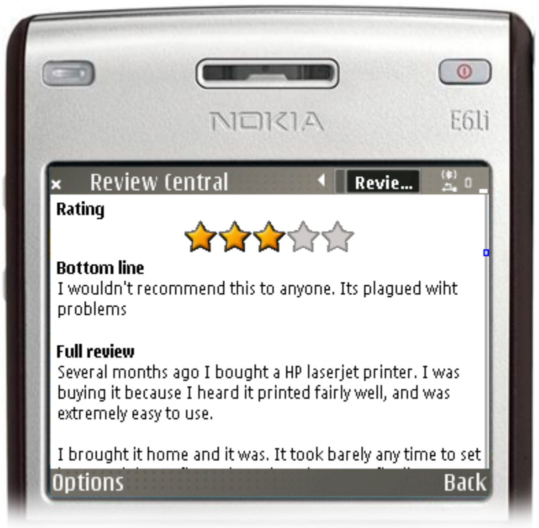
for home use? the only choice
Rated 5 out of 5

Great quality, performance, and valu...
Rated 5 out of 5

Just make sure you raise a litter
Rated 3 out of 5

Now This Is A Printer!
Rated 5 out of 5

Options Close applet


Review Central Revie...

Rating

Bottom line
I wouldn't recommend this to anyone. Its plagued wiht problems

Full review
Several months ago I bought a HP laserjet printer. I was buying it because I heard it printed fairly well, and was extremely easy to use.

I brought it home and it was. It took barely any time to set

Options Back

(b) Exclusive perspective showing “overview” view.

(c) Exclusive perspective showing “details” view.

Figure 4.7.: Example applet implementing a product review service.

The main functionality of the example applet is illustrated in Figure 4.7. The applet consists of the following files:

- `scripts/main.lua` contains the Lua callback functions.
- `views/overview.bit` contains the BITML code for a view named “overview” (see Figure 4.7(b)).
- `views/details.bit` contains the BITML code for a view named “details” (see Figure 4.7(c)).

- Several image files in the `resources` directory.

The main script file is shown in Listing 4.7.10. If the applet is part of the active runlist, the browser invokes the `run_aggregation_mode` function whenever the user scans a tagged object in the aggregation perspective. In this function, the applet sends two queries to a single resource repository. The first query is for resource descriptions that match the tagged object's tag id as well as the "score" profile. The second query is for resource descriptions that again match the object's tag id, but have their profile set to "review". If the first query yields a result with at least one resource description (line 18), the content of the first resource description's "data" field is extracted. It contains a numerical rating of the product. The second query returns a list of resource descriptions that each contains a textual product review. The number of resource descriptions (`#reviews`) is calculated and appended to the output.

As soon as the user selects the applet in the aggregation perspective (Figure 4.7(a)), the browser invokes the `start_exclusive_mode_coupled` function. This function again fetches all reviews from the resource repository and stores the resulting list of resource descriptions into a global variable named `review_resources` (line 38). The script then asks BIT to display the view named "overview", whose definition is shown in Listing 4.7.10.

The view iterates through all resource descriptions (line 5) and parses the JSON string in their "data" fields. In this example, resource descriptions following the "review" profile are expected to contain the review (i.e., the actual resource) in the "data" field. The JSON parser returns the review as a structured Lua table, which is saved into the global `reviews` array. For every review, a list entry showing the review's title and numerical rating is created (lines 7–10).

When the user selects a list item, the code in the `<menu_item>`'s `action` attribute (line 14) calls the `bit.show_view` function, asking the browser to show the view named "details". The function is given an assoviative array as a second argument. This associative array is accessible in the new view under the name `args`. The definition of the "details" view is reproduced in Listing 4.7.10 and shows how the various attributes of the review, which is available through the `review` key of the `args` associative array, are displayed.

Listing 4.1: Review applet's `main.lua` file.

```lua
REPOSITORY_URL = "http://resource-repository.example.com/"
PROFILE_REVIEW = "review"
PROFILE_SCORE = "score"

function run_aggregation_mode(tagged_object)

  local repository = bit.get_repository(REPOSITORY_URL)

  -- BIT uses associative arrays to pass a variable number of
  -- arguments to API functions. See API documentation for details.
  local score = repository.lookup_resource(
    {tag_id = tagged_object.tag_id, profile = PROFILE_SCORE})

  local reviews = repository.lookup_resource(
    {tag_id = tagged_object.tag_id, profile = PROFILE_REVIEW})

  local output = ""
  if #score > 0 then
    output = "Rated " .. score[1].data.value .. " out of 5"
  end

  if (#reviews > 0) or (output ~= "") then
    if output ~= "" then
      output = output .. ", "
    end
    output = output .. #reviews .. " reviews available"
  end

  if output ~= "" then
    bit.set_aggregation_image("icon.png")
    bit.set_aggregation_text(output)
  end

end

function start_exclusive_mode_coupled(tagged_object)
  local repository = bit.get_repository(REPOSITORY_URL)
  review_resources = repository.lookup_resource(
    {tag_id = tagged_object.tag_id, profile = PROFILE_REVIEW})
  bit.show_view("overview")
end

function start_exclusive_mode_uncoupled()
  -- This applet does not run in uncoupled state.
end
```

Listing 4.2: Review applet's `overview.bit` file.

```
<view title = 'ReviewCentral.com'>
  <list>
    <?
      reviews = {}
      for i = 1, #review_resources do
        reviews[i] = bit.json_decode(review_resources[i].data.value)
        bit.out('<list_item description="Rated ' ..
          reviews[i].rating .. ' out of 5">')
        bit.out(reviews[i].title)
        bit.out('</list_item>')
      end
    ?>
    <menu>
      <menu_item action="bit.show_view('details', {review = reviews[index]})">
        Show details
      </menu_item>
    </menu>
  </list>
</view>
```

4.8. Browser Architecture and Implementation

As a part of this thesis, a prototype of a browser for the Internet of Things was developed. In this section, we will first present the high-level architecture of the browser and its main components. After briefly discussing how applets are executed within the browser, we will also highlight some privacy issues and a few basic mechanisms to mitigate them. We will conclude this section with an overview of the actual implementation of BIT and the technologies used.

4.8.1. Components

As a runtime environment for applets, BIT acts as an additional layer between the mobile phone's operating system and the services that are available to users. BIT's role on the mobile phone platform, its interaction with the environment, as well as its main components are illustrated in Figure 4.8.

The functions of these components are as follows:

Resource discoverer The resource discoverer connects the browser to the open lookup infrastructure. It encapsulates the protocols used to access resource repositories and provides a local interface that allows both applets as well as browser components to find resources, such as information on tagged objects or services (i.e., Listing 4.3: Review applet's `details.bit` file.

```
<view title = "ReviewCentral.com" action="bit.show_view('overview', {})">
  <media>

    <text style="font-weight: bold">Pros</text>
    <text><? bit.out(args.review.pros) ?></text>
    <text/>

    <text style="font-weight: bold">Cons</text>
    <text><? bit.out(args.review.cons) ?></text>
    <text/>

    <text style="font-weight: bold">Rating</text>
    <image source="stars_<? bit.out(args.review.rating) ?>.png"/>

    <?
      if args.review.bottom_line ~= '' then
        bit.out('<text style="font-weight: bold">Bottom line</text>')
        bit.out('<text>' .. args.review.bottom_line .. '</text>')
        bit.out('<text/>')
      end
    ?>

    <text style="font-weight: bold">Full review</text>
    <text><? bit.out(args.review.full_review) ?></text>

  </media>
</view>
```

applets).

Applet manager The applet manager is responsible for the lifecycle of applets and coordinates their execution. Whenever a tagged object is recognized, it leverages the resource discoverer to find and start applets as described in Section 4.6.8. It also manages the installation of new applets and keeps track of runlist configurations as well as tied applets. In addition to the resource discoverer, the applet manager relies on the reader manager to obtain notifications on recognized tagged objects, the storage engine to load already installed applets, as well as the scripting runtime to execute applets.

Scripting runtime The scripting runtime provides the Lua interpreter that is required to execute applets. It also makes sure that the BIT API is available to applets at runtime.

BIT API The BIT API allows applets to use the functionality of resource discoverer, storage engine, reader manager, context grab-

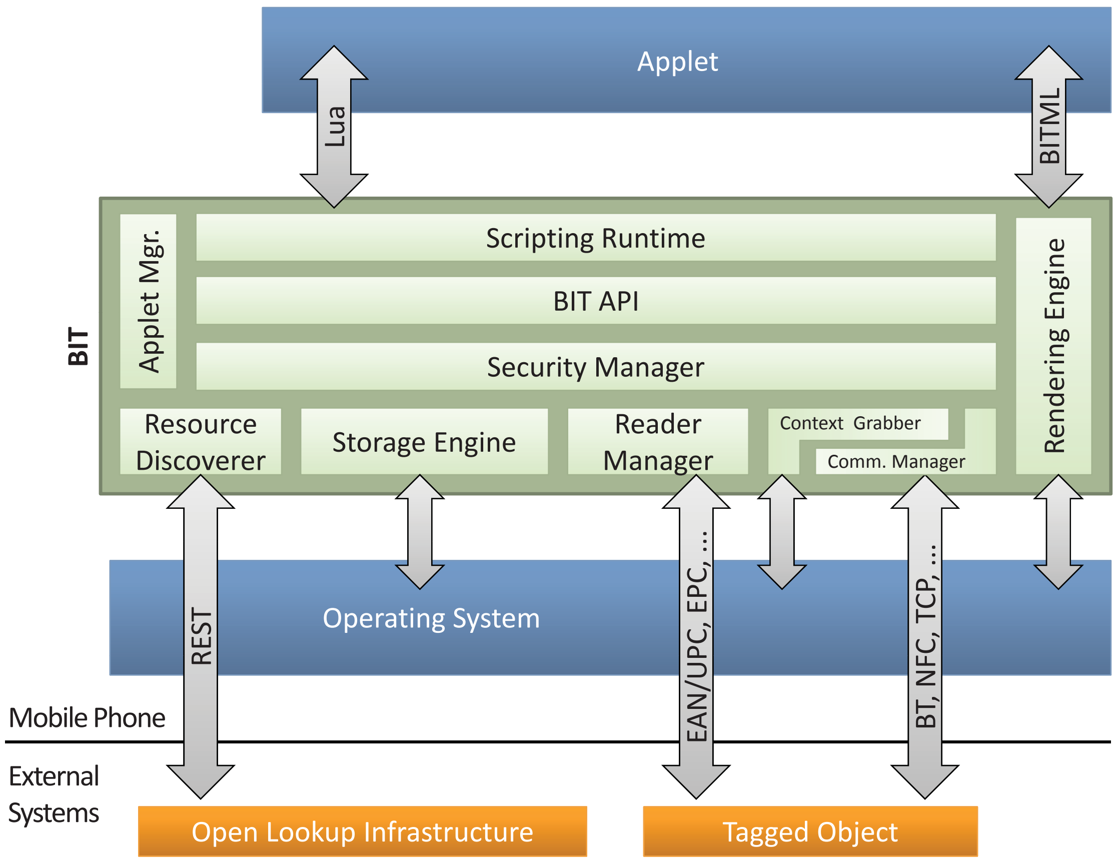


Figure 4.8.: Architecture of BIT (adapted from [36]).

ber, and rendering engine in a standard way that ensures portability of applets across different browser implementations and mobile device platforms.

Security manager The security manager controls an applet's access to the functionality provided through the BIT API. It makes sure, for example, that an applet can only access its own data stored in the browser. It also plays a role in protecting a user's privacy, which is further discussed in Section 4.8.3.

Rendering engine The rendering engine is invoked through the BIT API and allows an applet to present its user interface on the screen. It parses the BITML code of views, which are provided by an applet.

Storage engine The storage engine provides means for both other browser components and applets to persistently store data in the browser.

Reader manager The reader manager is used by the applet manager and, indirectly through the BIT API, applets to recognize tagged objects. It wraps the different tagging technologies that may be available on a particular mobile device (e.g., barcodes, NFC, EPC, etc.) and provides a simple interface that abstracts from the specific labeling standard used.

Communication manager This component is responsible for establishing and managing bidirectional connections with tagged objects that offer communication capabilities. It is used by the BIT API to provide the endpoint abstraction discussed in Section 4.6.9.

Context grabber The context grabber allows applets to acquire information about their context. This includes, for example, location and status information on the last recognized object. Some of its functionality is, together with that of the communication manager, exposed through the endpoint abstraction.

4.8.2. Implementation

Based on the above architecture, we implemented a prototype of our browser for Nokia S60, which is currently the most widely spread smartphone platform. The browser itself is written in Python, which we used because it allows for the rapid development of applications for S60 phones. Python is available on the S60 platform through the PyS60 project, which extends a Python port with some S60-specific functionality [133]. This includes the creation of user interfaces or access to the phone's communication capabilities, such as Bluetooth.

PyS60 allows developers to access *extensions*. An extension is a binary module that is written in the platform's native Symbian C++ language. We used this mechanism to import the BaToo Barcode Recognition Toolkit[7], which offers fast and robust barcode recognition using the phone's built-in camera.

The same approach, i.e., a Python extension module written in Symbian C++, was used for the Lua interpreter. Lua is not available in any form on the S60 platform. We therefore ported the source distribution of the Lua interpreter[8] along with Lunatic Python[9] to Symbian S60. Lunatic Python provides a "bridge" between the Python interpreter and the Lua interpreter. With the resulting S60 module, a PyS60 program,

[7]http://people.inf.ethz.ch/adelmanr/batoo/
[8]www.lua.org
[9]http://labix.org/lunatic-python

Figure 4.9.: Hardware platform (Nokia E61i) used for the prototype implementation of BIT.

such as BIT, can import and control a Lua interpreter from within Python.

The hardware platform that we used was a Nokia E61i mobile phone (see Figure 4.9). This phone features WLAN, Bluetooth, and a built-in camera, and it has a screen resolution of 320 × 240 pixels. It does, however, not provide an NFC module. Unfortunately, there is currently no device available that offers NFC on the S60 platform. NFC-enabled phones are based on the S40 platform, which can only be programmed in Java ME, and are not considered "smartphones" or "feature phones", as they are considerably less powerful.

The particular phone that we used for our prototype had been modified by Nokia Research Center [132]. Its standard battery cover had been replaced with a somewhat bigger one that contains an UHF RFID reader based on EPCglobal's Class 1 Generation 2 protocol [31], which is today's most commonly used RFID standard in logistics. The reader can be controlled through a custom UDP-based protocol.

4.8.3. Security and privacy considerations

As applets contain Lua scripts, they represent mobile code, which poses some risks to its host [128]. BIT mitigates these risks through the use of the Lua interpreter as a sandbox. As a virtual machine that lies above the operating system, the scripting runtime may offer some degree of protection for the host [21]. Access to operating system functionality is possible only through the BIT API and can be controlled by the security manager. Applets themselves are isolated from each other. Every applet has its own state in the Lua interpreter, which is unavailable during the execution of another applet.

The idea of using a single tool, a browser for the Internet of Things, to interact with all sorts of physical objects in any conceivable situation naturally raises the question of privacy. While a full analysis of the privacy implications of such a tool would be beyond the scope of this thesis, we will highlight some specific privacy aspects.

With our framework, one of the main reasons for privacy concerns is that all permanent applets which are part of an active runlist are notified about every single tagged object that the user scans. Whenever the user scans a tagged object, these applets are executed in aggregation mode and send a query to a remote resource repository. Similar to an HTTP cookie, an applet could easily include a token in its requests, which would allow the service provider to link consecutive requests even when the client's IP address changes. While the BIT framework inherently relies on all applets being notified about reads and their ability to communicate with remote hosts, this represents an alarming idea to many users. In effect, it would allow all service providers to gain detailed insight into a user's everyday scanning activities — no matter whether a specific service is actually used or not.

BIT mitigates this problem by limiting the information an applet can transmit when it runs in aggregation mode. In order to prevent applets from revealing any information that could be used to identify the user, the security manager blocks all communication operations bar resource repository lookups. As a result of this, applets can only use the *tag ID*, *profile*, and *context* fields in lookup requests to share data with the backend infrastructure. Since it would still be possible to encode an identifying token in, for example, a specially crafted profile name, these fields are futher filtered by the security manager:

- The *tag ID* field is always set to the tag ID that was last recognized by the reader manager.
- For the *profile* field, only values from a list of known profiles are acceptable. This list is distributed with the browser, but can be extended as the browser is used.
- Like the *profile* field, the *context* field can contain well-known context types only. In addition, only a context element's name,

but not its value can be specified. Comparable to the *tag ID*, the context value is automatically set to the value provided by the context grabber.

Note that these restrictions apply in aggregation mode only. In exclusive mode, an applet can communicate freely because, unlike in aggregation mode, it is only notified about tagged objects that are recognized while the user explicitly uses the service.

However, another problem arises as a consequence of this. Instead of immediately transmitting tag reads to a remote host, an applet could log them to the storage service while running in aggregation mode. When it is later started in exclusive mode with full privileges, it could read this log and share it with a backend server. For this reason, the security manager also blocks write operations to the storage manager while the applet runs in aggregation mode. Read operations are still possible and necessary (e.g., for an allergy checker to fetch a user's dietary restrictions).

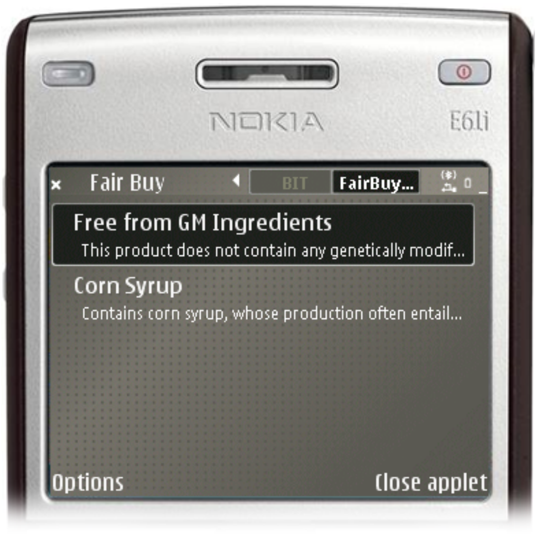
4.9. Example Services and Discussion

In order to validate both our framework design as well as our prototypical browser implementation, we created nine example services for BIT. Our goal was to assess the practical value that our framework can provide for relevant services and to illustrate the range of different service types that are supported. In keeping with this objective, we mostly selected services that had been proposed or taken up and refined by others [3, 143, 25, 28, 119, 140].

This section will start by presenting these applets and reviewing their functionality. We will then discuss the experiences we made in the process of their implementation. In particular, we aim to shed light on both the benefits and shortcomings of our framework when compared with a software development approach that is based on established and widely used tools for mobile platforms.

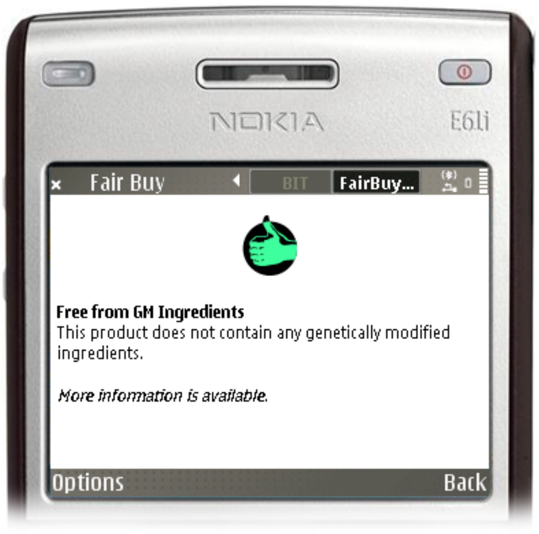
4.9.1. Product reviews

The first applet that we implemented provides a product review service. It allows users to browse other consumers' written opinions regarding a tagged product at hand. In addition, a numeric score for the product



Free from GM Ingredients
This product does not contain any genetically modif...
Corn Syrup
Contains corn syrup, whose production often entail...

(a) Overview with browser menu.



Free from GM Ingredients
This product does not contain any genetically modified ingredients.

More information is available.

(b) Details view.

Figure 4.10.: *Example applet.* Implementing a "political shopping" service.

is shown. The screenshots of this applet were already reproduced in Figure 4.7 on page 101, and we also listed its source code there.

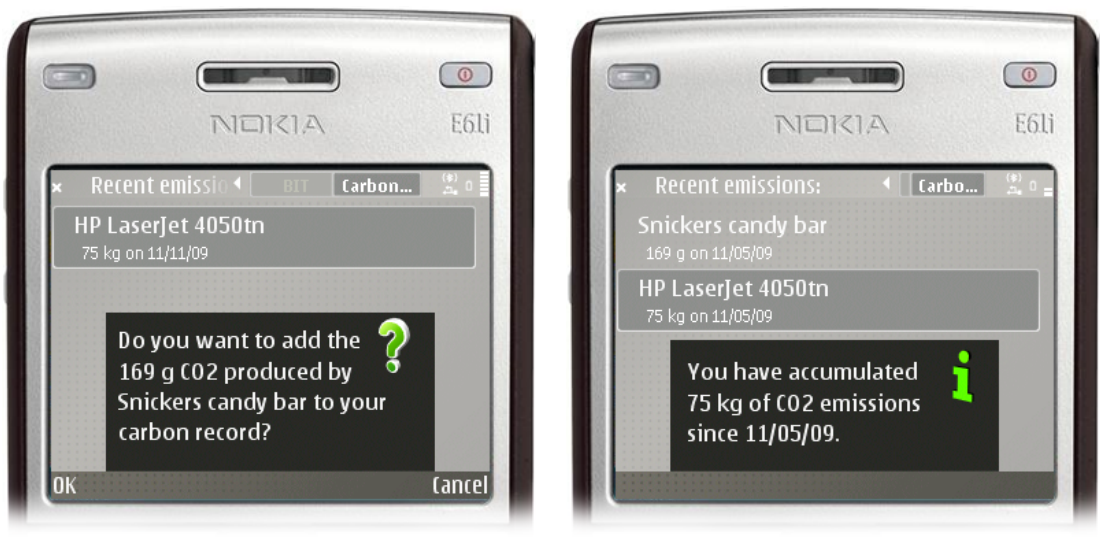
This example illustrates the use of mixed content in the `<media>` element. It also relies on the wrapper functionality offered by the open lookup infrastructure's resource repositories. In this example, we use a wrapper to calculate the average score of resources and another one to include reviews from Amazon.com, which represents an external data source.

4.9.2. Political shopping

This applet is similar to the product review applet (see Figure 4.9.2). The service in this example is provided by an imaginary consumer pressure group and illustrates how applets can be used for "political shopping" [66]. Information is again fetched from a resource repository, operated by the consumer pressure group. In addition to the overview displayed directly in BIT, the applet also links to articles on external web sites providing further background information.

4.9.3. Carbon footprint calculator

The carbon footprint calculator lets consumers keep track of the carbon emissions produced by their shopping activities. It is an example of a persuasive technology application [43] (sometimes also called "persua-



HP LaserJet 4050tn
75 kg on 11/11/09

Do you want to add the 169 g CO2 produced by Snickers candy bar to your carbon record?

Snickers candy bar
169 g on 11/05/09

HP LaserJet 4050tn
75 kg on 11/05/09

You have accumulated 75 kg of CO2 emissions since 11/05/09.

(a) Adding emissions to personal record. (b) Personal emissions record.

Figure 4.11.: *Example applet.* Implementing a carbon footprint calculator.

sive computing"), i.e., an interactive application which aims to change a person's attitudes and behavior.

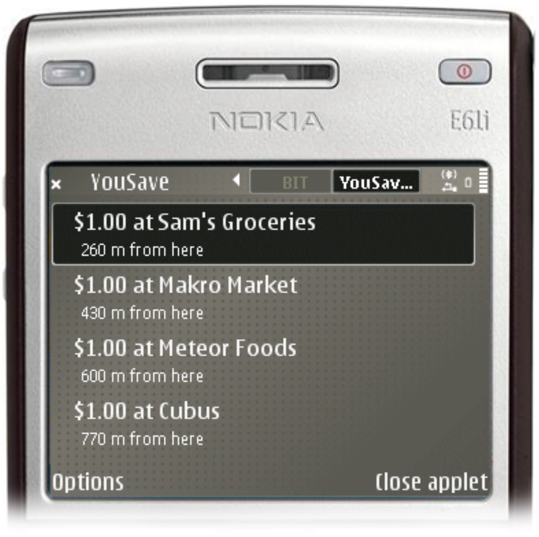
In the aggregation perspective, the applet shows users the carbon emissions produced by the product at hand (see Figure 4.2 on page 79). When the applet is selected and displayed in the exclusive perspective, it prompts users whether they want to add the product's emissions to their personal carbon record (Figure 4.11(a)). The exclusive perspective also shows a list of all entries in the user's personal record (ordered by date). Users can review their total carbon footprint (Figure 4.11(b)) and reset their personal record.

The carbon footprint calculator is an applet that relies on the browser's storage capabilities because it collects data that must be preserved for use in subsequent invocations.

4.9.4. Allergy checker

The allergy checker notifies users when a tagged product should not be consumed according to their dietary requirements. Users can configure these dietary requirements after manually starting the applet in exclusive mode. In order to manage these requirements, the applet again leverages the browser's storage engine to persist data.

The applet aims to not interfere with the user unless necessary. It does so by not producing any output in aggregation mode unless a tagged object does not match the user's dietary requirements. A screen-



$1.00 at Sam's Groceries
260 m from here
$1.00 at Makro Market
430 m from here
$1.00 at Meteor Foods
600 m from here
$1.00 at Cubus
770 m from here

Figure 4.12.: *Example applet.* Implementing a price comparison service.

shot of the applet in aggregation perspective is provided in Figure 4.2 on page 79.

4.9.5. Price comparison

"You Save" is an imaginary location-aware price comparison service. When the applet runs in aggregation mode, it queries the service provider's resource repository for cheaper offerings of the same product. The lookup request includes the mobile device's current geolocation. The most attractive alternative offering — depending on its price and location — is then shown in the aggregation perspective along with its distance, which is calculated by the applet. When the user selects the entry, the applet opens in the exclusive perspective and shows a list of potentially attractive offerings (see Figure 4.9.5). For every offering, address information and opening hours are displayed. If an entry is selected, BIT opens Google Maps in the mobile phone's standard web browser to provide users with detailed directions from their current location to the selected store.

4.9.6. Shopping list

This applet allows users to prepare a personal shopping list on the mobile phone by scanning the products that should be bought on one of the next shopping trips. It would typically be part of a "home" runlist. Upon every execution of the runlist, the applet creates an entry in the aggregation perspective (see Figure 4.7(a) on page 101).

When the user selects this entry, the applet shows a form consisting of a field for the product name and another one for the quantity that is to be bought. The name field is pre-filled with the product's name that is retrieved from the manufacturer's resource repository. Once the user saves the form, a reminder is created and added to the shopping list. The applet can also be started manually in exclusive mode to see all list entries. In this mode, entries can be edited or checked off, and the entire list can be cleared.

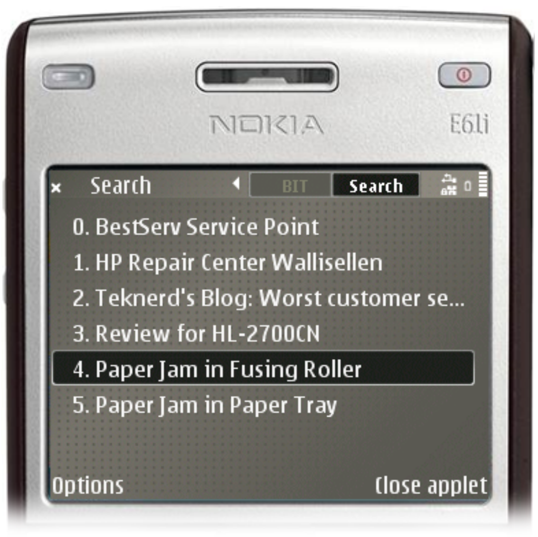
4.9.7. Search service

This example applet is based on the open lookup infrastructure's search service. We set up a search service that creates an index of standard web sites on the internet (see Section 3.4.3), which allows users to leverage BIT to also find relevant product-related information that is published on standard web sites.

When the search applet is part of the active runlist, it queries the search service whenever the user scans a tagged object. The query includes context elements that are obtained from the browser's context grabber, such as the current location or status information provided by the tagged object (if available). If resources are found, the applet creates a notification that is listed in the aggregation perspective (see Figure 4.7(a) on page 101). After selecting this entry, the applet is started in the exclusive perspective and shows a list of resources found by the open lookup infrastructure's search service (Figure 4.13(a)). The list is ordered by the search service based on the context elements that were provided by BIT, so as to show more relevant entries first. If a search result is selected, BIT opens the corresponding URL in the mobile phone's standard web browser. The example depicted in Figure 4.13(b) shows a web site with troubleshooting instructions that apply to a specific appliance condition. It was found by the search service via the device status that was provided by BIT.

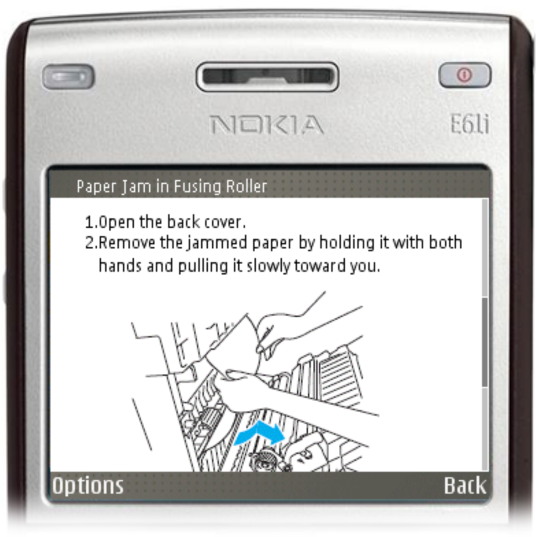
4.9.8. Coffee maker controller

Unlike the applets discussed so far, the remaining two examples are of the transitory instead of the permanent type. That is, rather than being installed by a user, these applets are started immediately without the user's explicit request when the corresponding tagged object is recognized.



0. BestServ Service Point
1. HP Repair Center Wallisellen
2. Teknerd's Blog: Worst customer se...
3. Review for HL-2700CN
4. Paper Jam in Fusing Roller
5. Paper Jam in Paper Tray

(a) Search results overview.

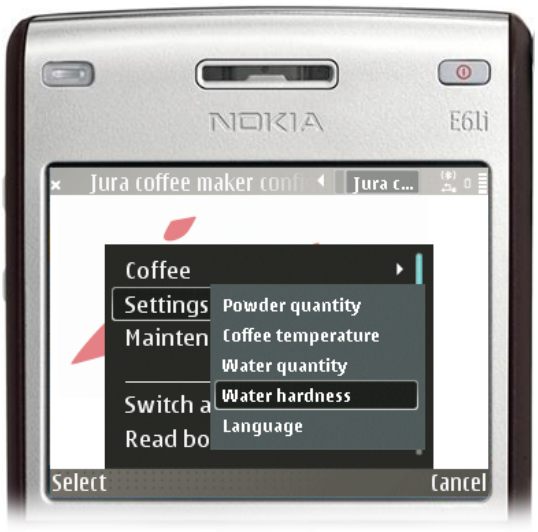


Paper Jam in Fusing Roller

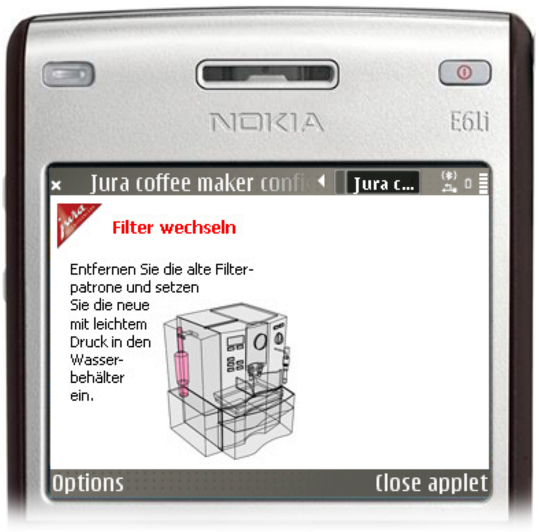
1.Open the back cover.
2.Remove the jammed paper by holding it with both hands and pulling it slowly toward you.

(b) Troubleshooting instructions on a web page.

Figure 4.13.: *Example applet.* This example is based on the open lookup infrastructure's search service.



(a) Some of the functions offered.



Filter wechseln

Entfernen Sie die alte Filter-patrone und setzen Sie die neue mit leichtem Druck in den Wasser-behälter ein.

(b) Instructions for changing filter.

Figure 4.14.: *Example applet.* Implementing a coffee maker controller service.

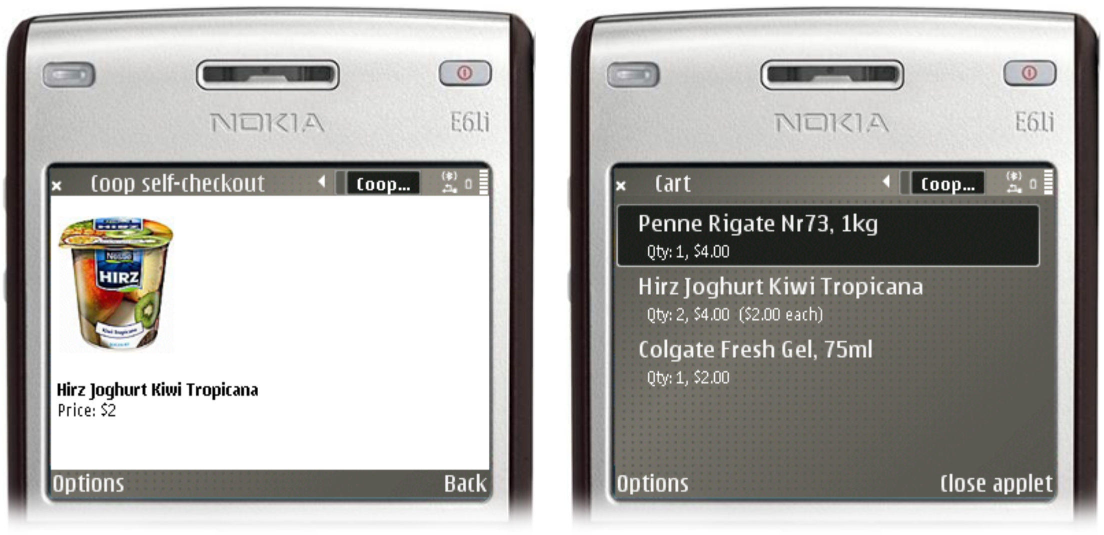
We created a "coffee maker controller", which allows the user to operate the appliance through BIT. The use cases we implemented are identical with the ones we presented in the user study of Chapter 2. In effect, this applet represents a BIT-based re-implementation of the AID, which we discussed in Section 2.5.4. Some of its functions are shown in Figure 4.14(a).

The controller applet relies on Bluetooth for bidirectional communication with the coffee maker. The applet itself does not contain any code for connection handling. It uses an endpoint as described above, which delegates all details regarding connection management to the browser. As we did not have a coffee maker with Bluetooth communication capabilities available, we implemented a "coffee maker emulator". This Python based application runs on a desktop PC, where it emulates the behavior and status changes of an actual coffee maker. It also shows a display that is updated according to the commands that are received from BIT over Bluetooth. The desktop application also simulates the coffee maker's status, which can by changed directly in the desktop application's user interface. For example, if the emulator is set to request a filter change, the coffee maker applet automatically shows a corresponding assistant as soon as it is started (see Figure 4.14(b)).

4.9.9. Self checkout

The last example that we implemented is a self checkout applet as it could be provided in a retail store. It is again a transitory applet because we picture shoppers scanning a dedicated "launch tag" as they enter a store, which starts the applet automatically. While the applet is active in the exclusive perspective, it immediately shows store-related content as soon as an object is recognized. When, however, the aggregation perspective is active, the applet is not notified about reads because, as a transitory applet, it is not part of any runlist. This allows users to easily switch back and forth between their own runlist, which provides diverse, "neutral" product information, and the store's applet, which provides store-centric product information.

Figure 4.15(a) shows an example of an overview page that is provided by the applet after a product is recognized. If users decide to put the item into their physical shopping basket, they can selected the applet's corresponding function that places the item into their virtual shopping cart (see Figure 4.15(b)). Once they have collected all items, users can proceed to a payment point in the store to scan a special "pay" barcode. This barcode signals the applet that the costumer is done and ready to pay. The "pay" barcode provides an endpoint to the specific payment point it is affixed to, which allows the applet to establish a connection with the cash terminal built into the payment point. Like in the previous example, we used Bluetooth to transmit the virtual cart's content to the terminal, which can then ask the customer to pay for the purchase.



Hirz Joghurt Kiwi Tropicana
Price: $2

Penne Rigate Nr73, 1kg
Qty: 1, $4.00
Hirz Joghurt Kiwi Tropicana
Qty: 2, $4.00 ($2.00 each)
Colgate Fresh Gel, 75ml
Qty: 1, $2.00

(a) Store-specific product information. (b) Shopping cart content.

Figure 4.15.: *Example applet.* Implementing a self checkout service.

Applet	**Views**	**Views LOC**	**Scripts LOC**	**Total LOC**
Product reviews	2	51	34	85
Political shopping	2	43	30	73
Carbon footprint calc.	1	26	124	150
Allergy checker	2	38	107	145
Price comparison	2	33	69	102
Shopping list	2	38	88	126
Search service	1	24	29	53
Coffe maker controller	13	216	44	260
Self checkout	4	60	98	158

Table 4.3.: *Example applets metrics:* number of views, source lines of code (LOC) for views, LOC for script files, and LOC total.

4.9.10. Discussion

As shown by the nine example applets presented above, BIT supports the development of a broad range of different services for tagged objects. From a developer's perspective, the above examples also illustrate the framework's main benefit: A considerable reduction of development effort. Table 4.9.10 shows how many views and how many source lines of code[10] were needed to implement the services as described.

For illustrative purposes, we compared some of these numbers with

[10]We counted physical lines of code by using the UNIX `wc -l` command.

those of a comparable applet we had implemented earlier during the work presented in Chapters 2 of this thesis. The Java ME-based coffee maker controller that we built for the usability evaluation offers the same functionality as the example applet discussed above. The only difference between the two lies in their ability to communicate with a coffee maker. While the Java ME MIDlet does not communicate with a remote appliance, the BIT applet retrieves (and updates) the coffee maker's settings from our emulator using an endpoint provided by the framework. Despite its more limited functionality, the MIDlet's implementation required 931 source lines of code, which is considerably more than the 260 lines required by the BIT applet.

Lines of code are certainly not the most accurate metrics, and it is to be expected that a scripting language like Lua requires fewer lines of code than Java. Even so, these numbers give an impression of how developing a service for the BIT framework differs from creating the same service for another platform such as Java ME, which is a practical alternative that is widely used and considered to be relatively easy.

In Section 4.4, we set out the requirements that our framework would need to fulfill. In the remainder of this section, we will analyze how BIT meets these requirements.

Navigation of services BIT satisfies this requirement by providing an aggregation perspective that gives users an overview of relevant services. At the same time, several applets can run in the exclusive perspective at once. Users can easily switch back and forth between the applets in the exclusive perspective.

Responsiveness Service execution and switching between different services occurs sufficiently fast. However, BIT suffers from the shortcomings of Python for S60, which has a number of problems in its networking modules. The latency of requests varies greatly and is dependent on the network's characteristics in an often unpredictable manner. Due to these issues, the current prototype fails to be responsive at all times — even under very light load. A solution to this problem would consist in implementing BIT directly in Symbian C++ (or another language) rather than Python for S60.

Coordination of services Runlists are designed to execute a number of services at the same time and to distribute tag reads to applets.

Use without physical object BIT provides both a bookmark and history functionality, which allows users to access an object's services in its physical absence.

Service invocation types BIT meets this requirement by offering transitory and permanent applets as well as the possibility to tie applets to tagged objects.

Discovery Services themselves can be linked to a tagged object through the open lookup infrastructure. It is thus possible to discover previously unknown services by interacting with the object.

Appliance status recognition An appliance can advertise its status, which is made available to the applet through the `tagged_object`'s `context` element.

Appliance control This requirement is addressed by endpoints, which allow developers to transmit data (e.g., commands) to a remote tagged object without dealing with the actual underlying communication technology.

Acquisition of context information Such information is available through the BIT API.

Integration with backend infrastructure The BIT API provides functions that allow applets to access the open lookup infrastructure.

Persistent storage Applets can persistently store data via the BIT API.

Tagging technology-agnostic BIT meets this requirement by letting applets implement callback functions that are used to deliver notifications about tag reads in a technology-independent way through the `tagged_object` argument.

Platform and device independence As applets run entirely inside the browser, which provides its own user interface markup language as well as its own API, applets can be implemented in a platform- and device-independent way.

Easy-to-learn and rapid service development BIT borrows some of its ideas from web application frameworks. These include the mix of scipting and user interface markup code as well as the MVC-inspired distinction between views and controllers. Both BITML as well as the BIT API are lean, and Lua is similar to many other scripting languages. A developer who has some experience with web application development should be able to write BIT applets in a very short time. Finally, as we saw above, BIT applets are small and can be implemented quickly.

Design for low attention Runlists are designed to hide those applets from the aggregation perspective that do not produce any output. They also provide a means to prioritize more important applets, which allows users to ensure they do not miss critical information. Finally, the navigation to access the features of BIT is laid out so as to facilitate its use in a low-attention environment. Frequently used functionality (e.g., switching between exclusive perspective and aggregation perspective, turning on reader devices, etc.) can be accessed with one or two easy-to-remember keystrokes that do not depend on the applet's implementation.

Unified user experience The browser itself is responsible for the main user interface, which allows it to ensure a consistent appearance and navigation of services. Functionality that is not unique to an applet (e.g., closing the applet, turning on reading devices, etc.) is always accessible in the same way.

Security and privacy Each applet runs in its own sandbox, which mitigates threats to both the mobile phone platform as well as other applets. To offer some degree of privacy protection, BIT's security manager limits the ability of applets to share potentially sensitive data.

A weakness of the current prototype is its limited support for designing custom user interfaces. For example, it does not implement the full CSS specification, but provides support for a few basic features only to format and align text elements. This is due to two factors: On the one hand, Python for S60 does not offer all user interface features that are available on the S60 platform. Among other issues, the absence of certain UI widgets makes it very difficult to offer the creation of more flexible and appealing user interfaces through BITML. Again, this problem could be mitigated by implementing the prototype based on alternative tools rather than Python for S60.

Another question is whether BITML in combination with CSS is expressive enough to meet the needs of all service providers. For example, BITML is geared towards the traditional mobile phone form factor consisting of a simple display and a separate keypad. It does not specifically support touchscreen devices and the additional interaction methods offered by them (e.g., dragging elements on the screen). However, the focus of this thesis was not on the development of an abstract user interface description language. There is already much work in this area. The main interest of this thesis was to develop a general framework for the integration of many diverse services from different sources and how they can be deployed and executed in a usable way. We see existing work in the user interface domain as complementary, as it can be leveraged in future implementations to extend BIT's basic markup language.

An important benefit from the use of an interpreted scripting language is the platform independence of BIT. This can significantly reduce a developer's effort because the same applet can run on all mobile phone platforms for which BIT is available. However, this approach may have a drawback for some service providers. Since applets are distributed in source code form, it is easy for anyone to look into their implementation, borrow ideas, or create derivatives. Appliance manufacturers, for example, effectively disclose the interfaces to their physical products by offering a controller applet. With this information, it is not difficult to create a replacement for the manufacturer's controller, which may be perceived as a threat to their brand image by some. On the other hand this transparency is not unique to BIT. The bustling Web 2.0 community makes heavy use of technologies that disclose source code, which allows developers to learn from each other and build new, innovative mashups.

Ultimately, however, the nature of most services for tagged objects is different from many traditional (web) applications. Services tend to add to the value of a physical product, rather than being a sellable product themselves (e.g., a controller for a coffee maker). And in those cases where a service can be marketed without an associated physical product (e.g., a price comparison service), the service's real asset is not the applet, but its data collection. For these reasons, it seems acceptable that the source code of BIT applets is not further protected.

4.10. Related Work

The need to make services available on mobile devices has been addressed in several research projects that are related to our BIT framework. In this section, we will look into these projects and contrast them with our work. Although boundaries are not always clear-cut, our survey will touch on subjects such as user interface adaptation, smart environment control, and information access.

4.10.1. User interface adaptation

A popular approach to the problem of providing services that can be accessed from a range of different mobile device platforms has been to run the service in the backend infrastructure and ship its user interface to the mobile device, where it is adapted to the platform's capabilities.

Perhaps one of the most often cited system in this area is UIML, the User Interface Markup Language, by Abrams et al. [2] UIML is XML-based and allows user interfaces to be specified using abstract interface elements. These abstract interface elements are then mapped to platform-specific widgets using stylesheets. UIML is purely declarative and does not include a procedural scripting language. However, a user interface can define events. Events can be used to update the values of user interface attributes or invoke backend methods.

A similar system, also relying on abstract XML-based user interface descriptions and stylesheets, is proposed by Müller et al. [95]. However, the goal of this systems is to minimize the device's workload by transforming the abstract user interface into a concrete one in the backend.

Mori et al.'s TERESA and CTTE tools [92, 93, 94] are again based on the idea of abstract user interfaces from which a concrete interface is generated for a particular device. Their approach, however, introduces explicit task models from which abstract user interfaces are derived. A graphical tool is provided that assists developers throughout the modeling phase in the specification of the task model.

An agent-based system is presented by Mitrovic and Mena [91]. In their system, user interfaces are specified in XUL [52]. A suitable agent "travels" to the mobile device for rendering the user interface. For user input handling, a Java class must be written by the developer, which is then deployed onto the client device.

Nylander et al. [109] propose another device-independent language for describing service interaction. In their system, the final presentation, which is rendered by an interaction engine named Ubiquitous Interactor (UBI), can be adapted through the use of "customization forms" that map abstract "interaction acts" to platform-specific GUI widgets.

The PERCI (PERvasive ServiCe Interaction) project by Broll et al. [16] relies on Web Services to automatically generate user interfaces for mobile phones. PERCI transforms Web Service descriptions into abstract user interfaces that are then rendered on the mobile phone by a Java ME client or further transformed to HTML pages that can be viewed in the phone's browser.

Finally, compared to the above projects, the W3C's XForms Recommendation (i.e., standard) [13] is fairly simple, but also less powerful. It allows for the definition of forms in a platform-independent way. In XForms, a form consists of two parts: A data model that specifies the data elements that are collected in the form as well as a user interface that describes how these elements are collected. An XForm user interface can be described by means of a set of device-neutral, platform-independent form controls. Every data element from the model is bound to an instance of a form control. When the form is rendered, it is up to the browser to choose a concrete UI widget that is most suitable to present the control on the specific platform. The appearance of forms can be specified using CSS.

While the focus of our work was never on the development of an abstract user interface description language, these projects are similar to BIT in that they provide user interfaces that can be rendered on a number of of different mobile devices. Like these projects, the interaction elements provided by BITML are not specific to a certain platform, and the user interface's appearance is separated from its functionality.

Even though some of the projects discussed are designed to also support the speech modality, they mostly concentrate on graphical user interfaces. New interaction techniques, such as scanning with a barcode reader or touching an RFID tag, are not supported. Similarly, adaptation of a service based on the user's momentary context is not considered by these solutions. Unlike in BIT, device-specific features that are needed to gather context information are not accessible.

A drawback of purely XML-based approaches is the difficulty of implementing even basic application logic. Some projects, such as the Ubiquitous Interactor, delegate this task to the backend infrastructure. This comes at the cost of frequent request/response cycles, which may impair user experience. Others, such as UIML, provide a solution based on specifying basic application logic in a custom XML-based format. However, such declarative approaches can be much more cumbersome for developers than general-purpose procedural languages. Unlike the projects presented above, BIT offers a full-fledged scripting language, which makes it relatively easy to glue together user interface elements based on application logic.

Overall, the scenarios that we target with BIT require functionality that cannot be provided by existing work in the user interface adaption domain alone. At the same time, such projects have proposed standards for the description of user interfaces that are more detailed than BITML. Given the comprehensive body of work that exists in this area, we decided to keep our own markup language fairly simple. Future versions of BIT, however, could draw on these complementary efforts (e.g., XForms).

4.10.2. Smart environment control

One of our framework's goals is to support the development of services for controlling physical appliances. In this section, we will review other projects that leverage personal mobile devices to control individual appliances or a range of services embedded in a smart environment.

Olsen et al.'s XWeb [110] is a system that allows users to control all sorts of physical appliances with a single device. The assumption behind XWeb is that it is not a good solution to have a unique piece of software for every appliance. XWeb therefore introduces XTP, a "universal interactive service protocol", that can be implemented by appliances to ensure interoperability with any XTP-compliant client. XTP builds on HTTP and allows clients to control a service by inserting, updating, and deleting data in a tree that is made available by the service using the URL notation — similar to what is called REST today. User interfaces are designed using XViews, XML documents that define which nodes of the tree should be presented to the user for manipulation.

Within the Pebbles project, Nichols et al. developed the personal universal controller (PUC) [105], which is very similar to XWeb. It also introduces a communication protocol that appliances must support in order to interoperate with PUC, and it also uses abstract user interface descriptions that are rendered on the mobile device.

Comparing to both XWeb and PUC, BIT does not rely on a specific protocol that must be adopted by appliance manufacturers. As manufacturers may not be willing to support such a protocol, BIT's ability to automatically download an applet for an appliance has the benefit of allowing for the implementation of custom protocols.

In [15], Broll et al. present Collect&Drop, a technique to interact with services that is based on the idea of physically touching a number of NFC tags. A typical use case example consists in a movie poster that allows users to order tickets for a certain show of the movie by touching NFC tags that represent show times and number of tickets. When the user first touches an NFC tag, an "action item" is retrieved from the tag, which specifies all other items (e.g., show time and number of tickets) that need to be "collected" by the user before a service (e.g., a ticket vending service) can be invoked. Collect&Drop keeps track of all tags that are then touched by the user. Once all elements have been collected, the service is invoked by opening its URL, which is encoded in the action item, and passing in the collected items as arguments. As a result, the service can return a new element (e.g., the movie ticket) that is collected and stored for future use (e.g., invocation of a ticket validation service at the movie theater). In the BIT framework, the same functionality can be implemented with an applet. The applet's script can require the user to perform the same actions that are described in a declarative manner in Collect&Drop's action items.

Other projects have emphasized the systems perspective more than the interaction aspects. An example is the Universal Information Appliance (UIA) by Eustice et al. [37], which aims to be able to interact with any application and operate any electronic device in the user's environment. The UIA uses its own XML-based language MODAL (Mobile Document Application Language) to deploy applications on the mobile device. In some ways, MODAL is similar to UIML. However, it uses a tuplespace system as a communication middleware for the exchange of messages with the environment. This is an additional infrastructure requirement, and it remains unclear who would be willing to provide it at a large scale.

AlfredO by Rellermeyer et al. [118] is a lightweight middleware architecture that aims to enable modular applications that can be dynamically distributed among participating devices, such as touchscreens or vending machines. AlfredO builds upon the R-OSGI middleware [117], which extends the industry-standard OSGi specification to support distributed module management. AlfredO applications are structured in a presentation tier, a logic tier, and a data tier. The presentation tier resides on the mobile device, the data tier on the device to be controlled, and the logic tier can be dynamically assigned to either device. Input and output capabilities that are used by a user interface are modeled as OSGi services (e.g., "PointingDevice" and "ScreenDevice"). Due to the underlying R-OSGI middleware, AlfredO can offer a complete infrastructure for more complex applications. This requires an understanding of concepts that may not be familiar to a casual software developer. Thus, for the scenarios outlined in this thesis, BIT's simpler approach seems to be just as appropriate.

With similar applications in mind like Rellermeyer et al., Riekki et al. propose their REACHeS (Remotely Enabling and Controlling Heterogeneous Services) architecture [120, 131] that allows users to control public displays through their mobile phones. Unlike the other projects reviewed above, REACHeS is explicitly designed to be used with tagged objects, and in particular NFC. The system relies on URLs that are encoded in NFC tags, which are opened in the phone's web browser when the tag is recognized. This URL points to the central REACHeS server, which allocates a specific display that is encoded in the request URL and forwards the request to a server that actually implements the service to be shown on the display (e.g., a web site advertising a product). The service responds by updating both the display as well as the HTML page in the phone's browser, which contains a user interface that allows for controlling the display. REACHeS differs from the above projects in that it only uses technologies that are readily available on mobile phones, such as Ajax [50] and NFC's mechanism to automatically launch the browser when a tag is read. The system does not require any additional software on a standard NFC phone, and infrastructure requirements in the backend are modest. The downside of this simplicity is that REACHeS only works with NFC and HTTP over the phone's default data bearer, but does not support other tagging technologies or direct Bluetooth connections with local appliances.

All of the above projects concentrate on *using* services to control a smart environment and do not address the question of how users can *select* a service that matches their needs. In the case of NFC, for example, the service to launch is hard-coded in the tag. Others let users manually choose from a fixed list of services provided by an appliance. BIT aims to go beyond this by leveraging the open lookup infrastructure to dynamically bind services to tagged objects and supporting users with selecting the right service through its runlists.

4.10.3. Retrieving information

There are a number of projects concerned with retrieving information on a mobile device that are related to our framework. Some of these focus on providing information for tagged objects, others offer information for places, taking into account the user's context.

The Hewlett-Packard Labs' Cooltown [72, 71] is perhaps one of the most widely known projects in this field. Cooltown proposes an infrastructure based on world wide web technologies to create a "web presence" for people, places, and things. These entities can have their own web pages, where they can offer information and services. This web presence is advertised by infrared beacons that broadcast URLs (mainly used for places) or by encoding identifiers in barcode or RFID tags (mainly used for things). The web presence of an entity is managed by a "web presence manager" [27], a hyperlinked collection of web pages with information and services related to the entity. A web presence manager for a place, a "place manager", can also contain a resolver that maps the identifier encoded in a tag to a URL. As users enter a place, it is automatically recognized by the mobile device via infrared beacons or a "you are here" tag. When identifiers are sensed later, the place's resolver is used to look up the URL associated with the sensed entity. This allows for the context-dependent retrieval of information and services for a given entity.

Like BIT and our open lookup infrastructure, Cooltown envisions a universal interaction device to be used with any object and appliance that a user may encounter. However, while BIT supports places as a part of the context that can be supplied when looking up resources, Cooltown includes people and places as first-class entities in its architecture. For example, Cooltown allows users to see the momentary location of others. BIT does not offer such functionality because from the beginning, its focus was confined to tagged objects. Conversely, the Cooltown architecture is limited in its support of information and services provided by many independent parties. By default, identifiers are resolved using the local place manager. As a result, only those services that were registered by the owner of a place, who is in charge of the place manager, can be found. Services offered by third parties cannot be located. This problem is recognized in [70]. The suggested solution consists in information providers running their own web site with a resolver that is independent from the momentary place's resolver. However, this approach requires users to decide on the service they would like to use *before* scanning an object. Also, it may be necessary to try a long list of services before a relevant result is found. BIT frees users from making this decision by aggregating the services offered by several providers and ordering them according to the user's preferences. While the need to aggregate resolution services is mentioned in [70], it is not detailed how this could be done.

Finally, Cooltown relies on applications that are presented in a web browser. This approach can have significant security drawbacks. If an appliance, such as a coffee maker, is accessible only via the web, some form of authentication is needed in order to prevent unauthorized access. Since BIT does not rely on a web browser, it can leverage short-range communication technologies such as NFC and Bluetooth that naturally restrict access to users who are in the appliance's physical proximity.

The limitations of a web browser in the context of RFID-based mobile services are also discussed by Michael and Darianian [89]. Their focus is on striking a balance between thin-client solutions where the mobile phone and its web browser act merely as a presentation device for an application running on a remote server and rich-client solutions where the application executes directly on the mobile device. They propose a "quasi-rich client", a standalone application on the mobile terminal that fetches content from a server and displays it locally. This is the same approach that we also chose for BIT. However, BIT goes further by not only considering a single application, but a multitude of services. Many other aspects of the BIT framework (appliance control, context, presentation — just to name a few) are not addressed by [89].

With the PanOulu Luotsi, Kukka et al. [75] present a system that aggregates location-dependent information that can be accessed from a mobile phone. Data from different sources is integrated and mapped into the Luotsi's own data model — similar to wrappers in the open lookup infrastructure. Unlike BIT, however, the Luotsi system is centralized: all data is kept in a single database. As a result, new services cannot be deployed without the central database's support and cooperation.

Finally, the AURA (Advanced User Resource Annotation) system by Brush et al. [17, 137] allows users to view and share product-related annotations for products that are tagged with barcode labels. The use cases that are presented (e.g., product reviews, price comparison) are very similar to the examples that motivate our work. The paper also stresses the importance of an open architecture that allows any party to publish information and services. However, it does not propose the architecture of such a system, but presents the learnings from a user study that was carried out based on a prototype implementation.

4.11. Summary

When any party that is interested in providing information or services for a physical object can do so, the question arises how users can interact with the many offerings that may be available for a given object. Traditionally, there has been a single, isolated application on the mobile phone for every service. In this chapter, we presented the concepts, architecture, and implementation of a browser for the Internet of Things (BIT). BIT represents a *single point of interaction* that allows users to quickly gain an overview of available services and invoke those that seem interesting. It further frees users from the need to, one after another, start applications on the mobile phone that may offer content for a physical product — only to find out that none is available.

At the same time, BIT provides a software framework that significantly facilitates the creation of services for tagged objects. The framework offers a number of concepts and abstractions through its API that can be used by developers to implement services in a faster, easier, and more convenient way. Rather than needing to port a service to all major mobile platforms, each with its particular intricacies, a service needs to be written just once in order to run on any platform that offers a BIT implementation.

5. Conclusion

We began this thesis with the observation that the idea of augmenting everyday objects with digital services on mobile phones faces several obstacles and open issues. This is the case with respect to both the usability of such systems as well as their implementation. Our goal was therefore to identify factors in both fields that can help alleviate some of the current problems. In particular, we aimed to investigate, on the one hand, under which conditions users benefit from mobile devices to control physical appliances, and under which conditions they do not. On the other hand, we aimed to propose infrastructure components and a software framework that facilitate the development of mobile services for tagged objects.

In the remainder of this chapter, we will briefly revisit the arguments of this thesis.

5.1. Summary

We started this dissertation by reviewing some oft-cited applications of personal mobile devices to control physical appliances. We challenged the somewhat counterintuitive notion that it is more efficient to operate a physical appliance via a softkey-based user interface on a PDA than through the device's haptic controls, as suggested by previous studies. We argued that mobile devices are likely to offer benefits in special situations, such as a faulty appliance or a rarely occurring condition that requires the user's manual intervention. However, contrary to what is outlined in many scenarios in the ubiquitous computing literature, they cannot be expected to be of significant value during normal, everyday use of an appliance. In order to confirm or reject our claim, we formulated three hypotheses to be tested in a comprehensive usability evaluation:

- For controlling an appliance in exceptional situations, interaction based on a mobile phone is faster than interaction based on the traditional user interface.

- Looking up context-dependent information on the handling of an appliance is faster using a mobile phone than using traditional means (e.g., user manuals).
- To carry out everyday tasks, the use of an appliance's traditional user interface is faster than mobile phone-based interaction.

We continued with a description of our study setup. We used four typical appliances — a dishwasher, a coffee maker, a laser printer, and a radio — and 18 tasks. We distinguished between *control* tasks, *problem solving* tasks, *everyday* tasks, and *repeated control* tasks, the last representing the re-enactment of a control task that was performed before. Our study involved 23 participants and was based on a mobile phone running the software we had developed for the trial.

We then presented the results of our user study, in which we found support for all three hypotheses. Interestingly, mobile phone-mediated control remains significantly faster even after participants have familiarized themselves with an appliance. We concluded the chapter with a critical discussion of these results and pointed out the benefits arising from mobile phones that can sense an appliance's status and provide context-sensitive assistance for users.

In Chapter 3, we presented an *open lookup infrastructure.* We argued that, for two reasons, the increasing availability of information and services for tagged products requires a basic infrastructure for the publishing and discovering of such resources: First, it is needed by users to find relevant resources. Second, it frees developers from the need to implement anew a custom solution for a generic problem.

We stressed the importance for such an infrastructure to accommodate product-related resources from any party, including manufacturers, consumer interest groups, and even end-users. Our infrastructure consists of four core concepts: resources and their descriptions, resource repositories, a manufacturer resolver service, and a search service. Resource descriptions define a standard way for annotating and linking to resources, i.e., information or services. While resource repositories accommodate resource descriptions and allow for their context-aware lookup, the manufacturer resolver service can be used to find a product manufacturer's resource repository. The search service, finally, enables searches that span multiple repositories and makes standard web pages containing product-related information available as resources. We presented an implementation of the open lookup infrastructure as a proof of concept along with example applications that illustrate its applicability.

In Chapter 4, we proceeded by presenting BIT, a *browser for the Internet of Things.* We began by analyzing the obstacles that developers face when creating mobile services for tagged objects. Most notably, these include the need to create a separate implementation for every mobile phone platform and to deal with details (e.g., managing reader devices, connection handling, etc.) that are not a service's main concern. We also pointed out that users need a tool that coordinates the invocation of services on their behalf, or, in other words, a "single point of interaction" rather than many services to be invoked separately. We then sketched a scenario touching on many possible services, from which we derived the requirements as well as the design implications for our system.

Based on these findings, we detailed the core concepts of BIT and its software framework, the most important being the notion of applets. Applets are self-contained packages that are downloaded dynamically to provide services as users interact with tagged objects. We continued with a description of how services can be implemented using our framework. In particular, we presented BITML, the BIT markup language, which allows for the abstract, platform-neutral description of the user interface of services. We further introduced the BIT API, which offers abstractions that simplify service development and ensure platform independence. Finally, we described the integration of the Lua scripting language, which is used to glue the different applet components together.

These concepts were implemented in a prototypical browser. We presented its architecture, detailed the implementation on the Symbian S60 platform, and outlined our port of Lua to Symbian S60. We also pointed out some security and privacy aspects of BIT and the steps taken to mitigate them, which include sandboxing the Lua interpreter and limiting access to API functions.

To demonstrate the applicability of our system and its practical value for the development of services, we implemented nine example applets representing a variety of services. In particular, we used our framework to re-implement the Java MIDlet from our usability evaluation and showed that BIT significantly reduces the amount of code developers need to write. We critically discussed the strengths and limitations of our system. We found that it meets the majority of requirements set forth and our overall goal of easing service development, even though there is room for more sophisticated user interfaces. We concluded the chapter with a review of related research projects.

5.2. Contributions

The overall result of this dissertation is a spectrum of concepts, tools, techniques, and findings that facilitate the development of usable, mobile phone-based digital services for tagged objects. It provides insight into where the mobile phone as an interaction device can offer significant benefits for users, but also into the limits of the usability of this approach. It further provides an enabling infrastructure for such services on the one hand, and considerably simplifies the development process on the other hand.

In particular, the individual contributions can be summarized as follows:

- A *usability evaluation* of mobile phones to control physical appliances. We showed that, in exceptional situations, users achieve their goals significantly faster when interaction is mediated by a mobile phone. We also showed that in everyday situations, these benefits do not appear.

- An *open lookup infrastructure* for publishing and discovering information and services that are linked to tagged objects. The infrastructure is open to all interested parties, including product manufacturers, but also end-users.

- A *browser for the Internet of Things* (BIT) that coordinates the execution of the many services that may be associated with a tagged object and represents a "single point of interaction" for users.

- A *software framework* that minimizes the effort needed to create digital services for tagged objects and allows for the development of portable, platform-independent services. The framework integrates with BIT and offers a user interface markup language, an API, as well as a scripting language.

5.3. Limitations and Future Work

There are a number of open issues that could not be addressed in this thesis, but could be the subject of future work.

The current implementation of BIT is not always as responsive as it should be. This is a shortcoming of the Python for S60 module, which is used by BIT and occasionally introduces unpredictable network latency. This issue could be addressed by a native Symbian C++ implementation.

When applets are distributed, they are accompanied by their source code. As discussed in Section 4.9.10, this may not always be desirable. Some service providers may find the idea of BIT more attractive if it offered the option not to disclose implementation details.

We did not aim at creating a comprehensive abstract user interface description language for our framework. However, it would be interesting to investigate how such a language can achieve a balance between portability across different mobile platforms and interaction capabilities on the one hand and expressiveness on the other hand.

Tagged objects often do not have just information and services associated, but also documents, which can sometimes be personal. It would be helpful if such documents could as well be accessed by directly interacting with the physical object. This also leads to the question of how access to such documents can be managed, delegated, and shared in a usable way when ownership in a tagged object is transferred between or shared by several parties.

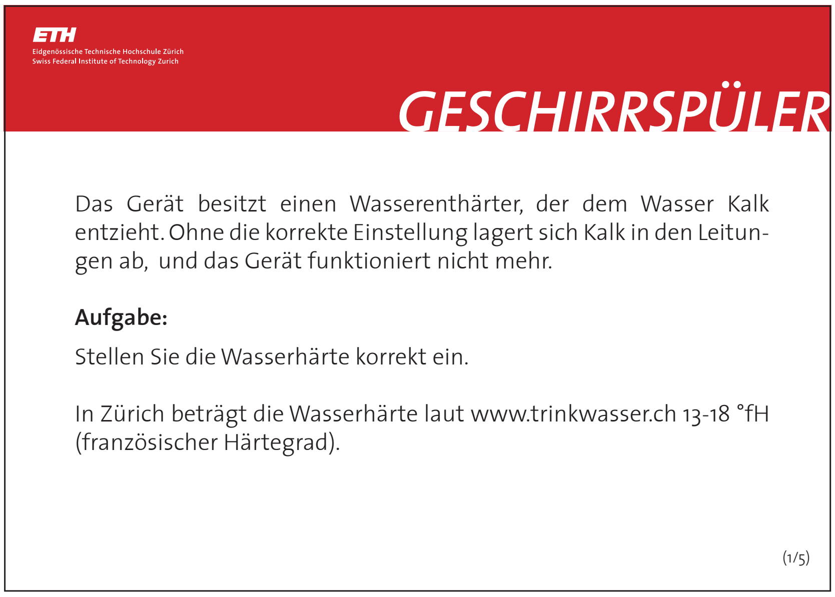
The selection of runlists is currently done manually. It could be worthwhile to combine BIT with research in the activity recognition domain to automatically select the appropriate runlist and to possibly also provide additional context for the search service.

Finally, a prototype of BIT has been implemented, but it would also be interesting to conduct a comprehensive study to gain insight in people's actual use of the system.

Appendices

A. User Study Task Assignments

For every task in our user study, users were given instructions written on a small card. This section shows the original text and layout of all cards used in the trial (reproduced from [135]).



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

GESCHIRRSPÜLER

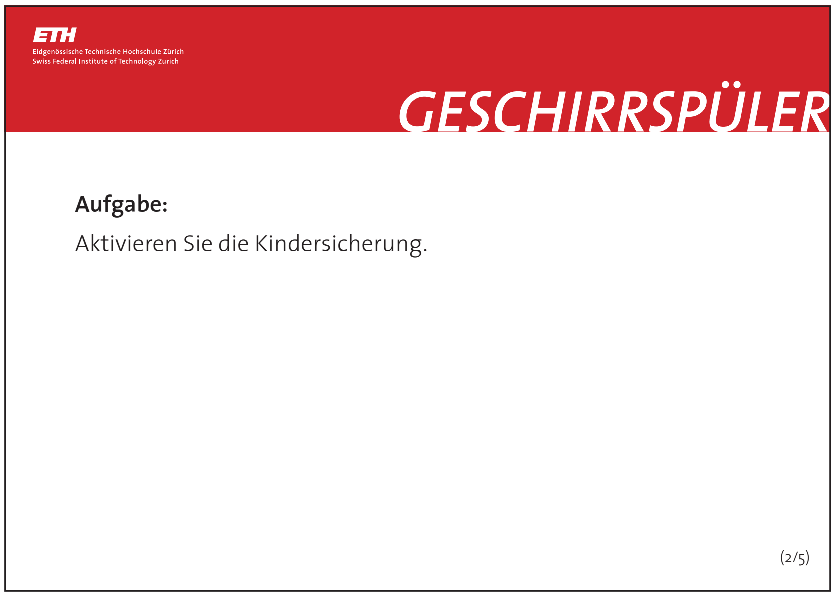
Das Gerät besitzt einen Wasserenthärter, der dem Wasser Kalk entzieht. Ohne die korrekte Einstellung lagert sich Kalk in den Leitungen ab, und das Gerät funktioniert nicht mehr.

Aufgabe:

Stellen Sie die Wasserhärte korrekt ein.

In Zürich beträgt die Wasserhärte laut www.trinkwasser.ch 13-18 °fH (französischer Härtegrad).

(1/5)



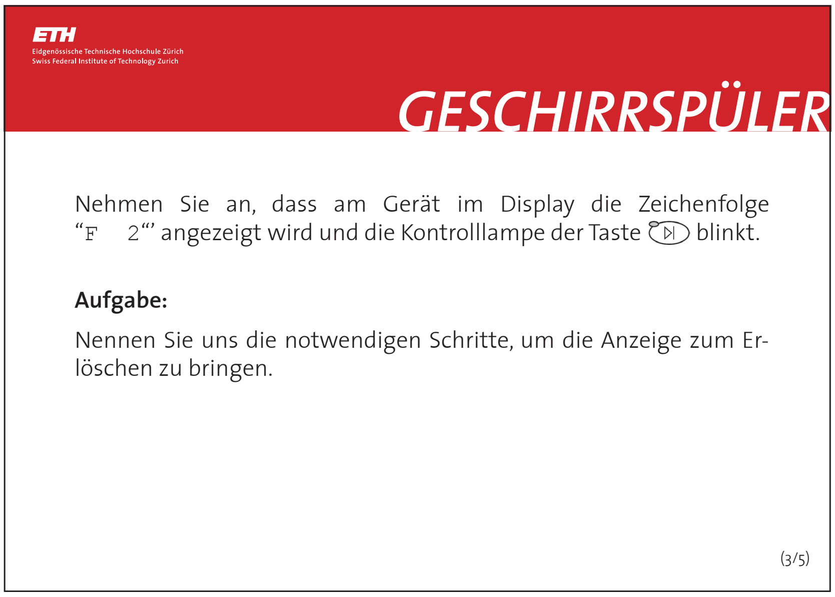
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

GESCHIRRSPÜLER

Aufgabe:

Aktivieren Sie die Kindersicherung.

(2/5)


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

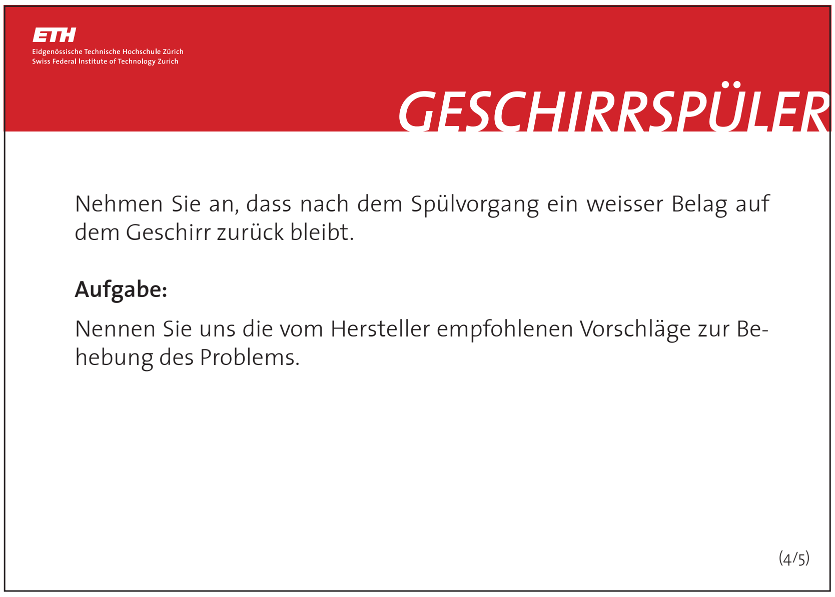
GESCHIRRSPÜLER

Nehmen Sie an, dass am Gerät im Display die Zeichenfolge “`F  2`“’ angezeigt wird und die Kontrolllampe der Taste ▷| blinkt.

Aufgabe:

Nennen Sie uns die notwendigen Schritte, um die Anzeige zum Erlöschen zu bringen.

(3/5)


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

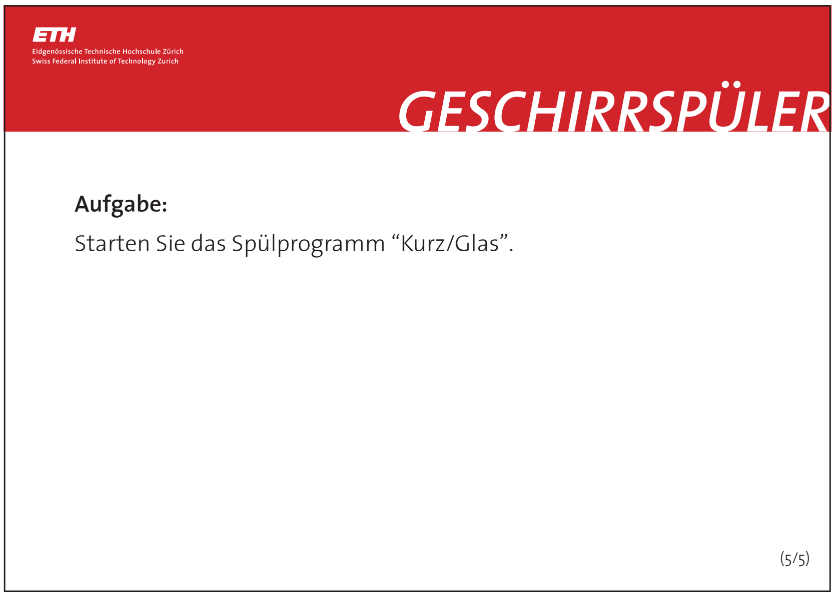
GESCHIRRSPÜLER

Nehmen Sie an, dass nach dem Spülvorgang ein weisser Belag auf dem Geschirr zurück bleibt.

Aufgabe:

Nennen Sie uns die vom Hersteller empfohlenen Vorschläge zur Behebung des Problems.

(4/5)

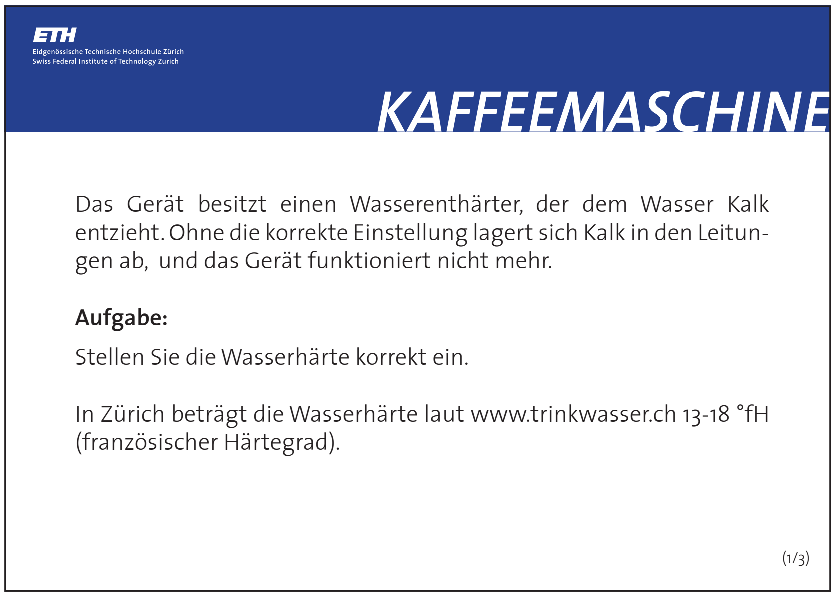

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

GESCHIRRSPÜLER

Aufgabe:

Starten Sie das Spülprogramm “Kurz/Glas”.

(5/5)



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

KAFFEEMASCHINE

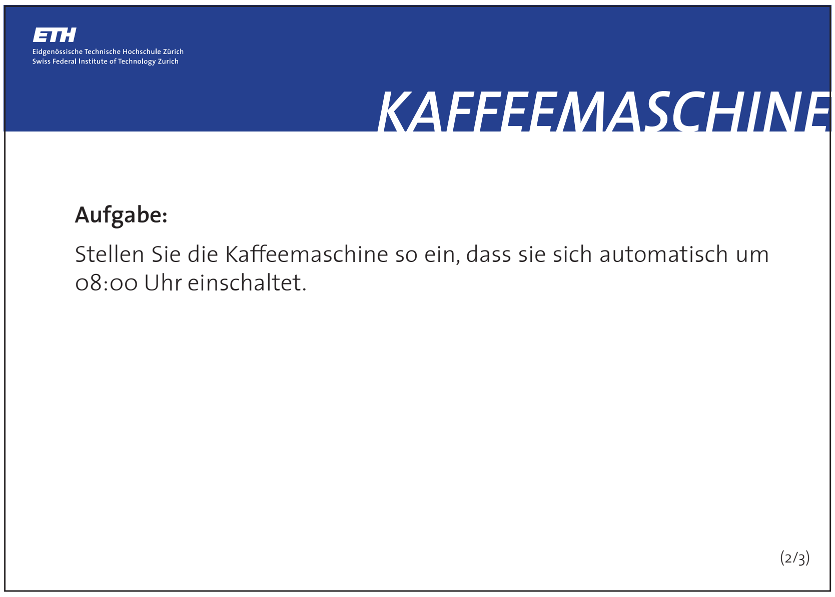
Das Gerät besitzt einen Wasserenthärter, der dem Wasser Kalk entzieht. Ohne die korrekte Einstellung lagert sich Kalk in den Leitungen ab, und das Gerät funktioniert nicht mehr.

Aufgabe:

Stellen Sie die Wasserhärte korrekt ein.

In Zürich beträgt die Wasserhärte laut www.trinkwasser.ch 13-18 °fH (französischer Härtegrad).

(1/3)



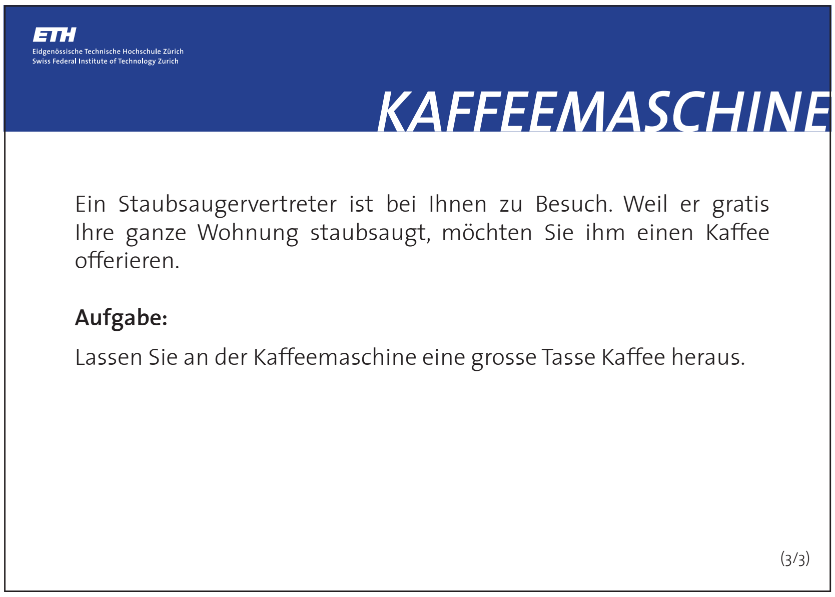
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

KAFFEEMASCHINE

Aufgabe:

Stellen Sie die Kaffeemaschine so ein, dass sie sich automatisch um 08:00 Uhr einschaltet.

(2/3)



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

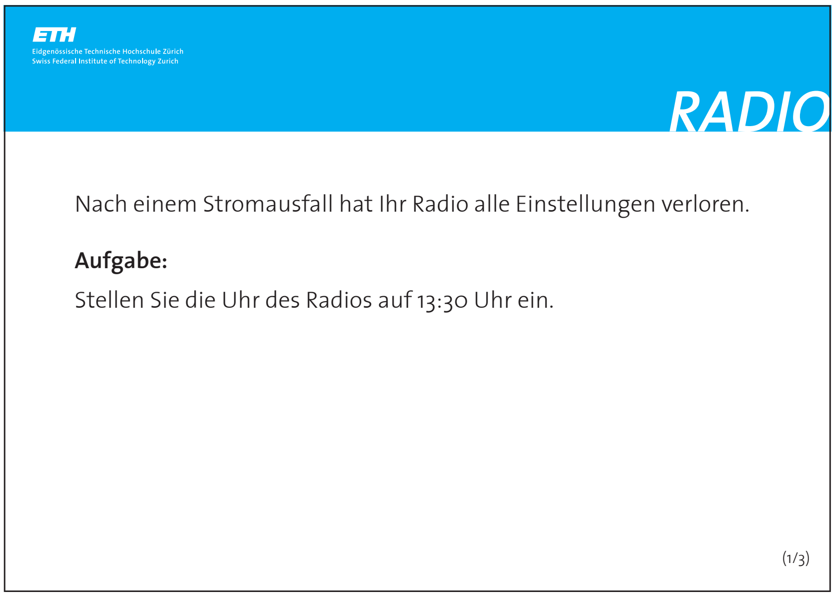
KAFFEEMASCHINE

Ein Staubsaugervertreter ist bei Ihnen zu Besuch. Weil er gratis Ihre ganze Wohnung staubsaugt, möchten Sie ihm einen Kaffee offerieren.

Aufgabe:

Lassen Sie an der Kaffeemaschine eine grosse Tasse Kaffee heraus.

(3/3)



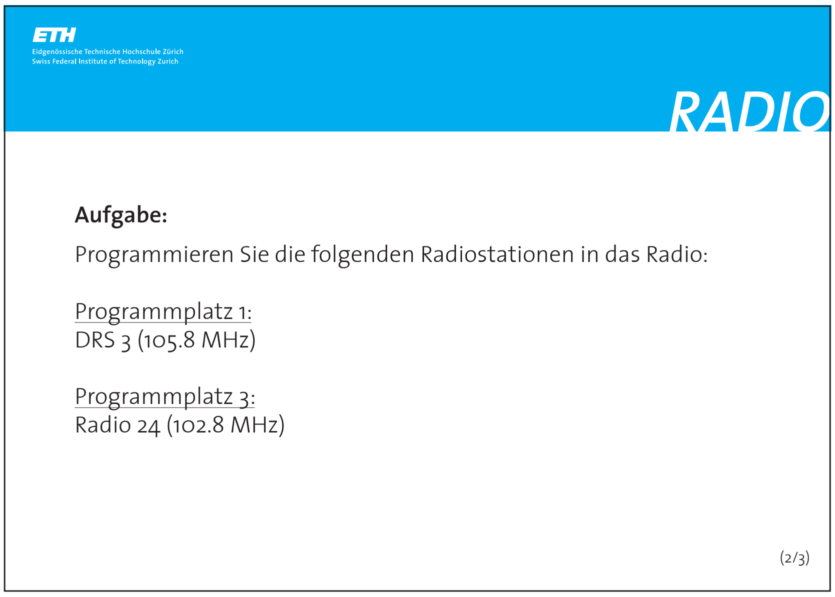
RADIO

Nach einem Stromausfall hat Ihr Radio alle Einstellungen verloren.

Aufgabe:

Stellen Sie die Uhr des Radios auf 13:30 Uhr ein.

(1/3)



RADIO

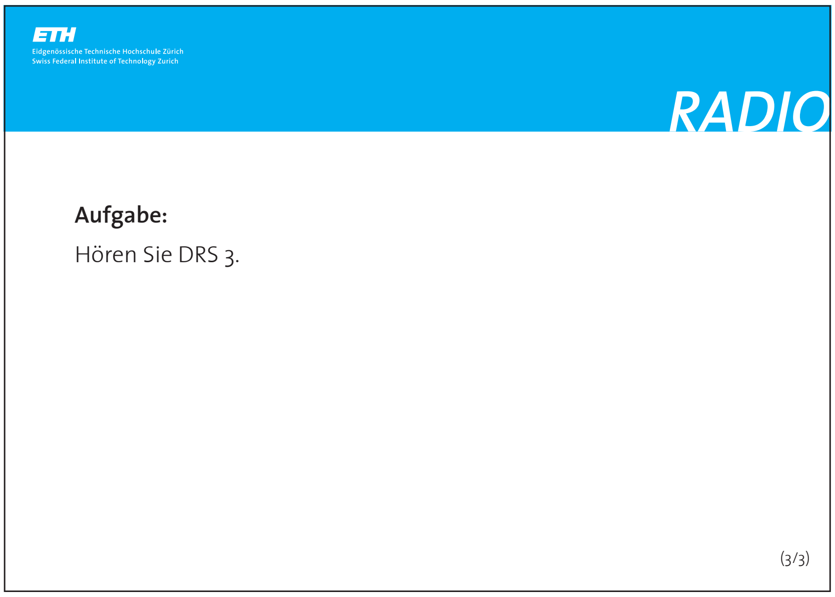
Aufgabe:

Programmieren Sie die folgenden Radiostationen in das Radio:

<u>Programmplatz 1:</u>
DRS 3 (105.8 MHz)

<u>Programmplatz 3:</u>
Radio 24 (102.8 MHz)

(2/3)

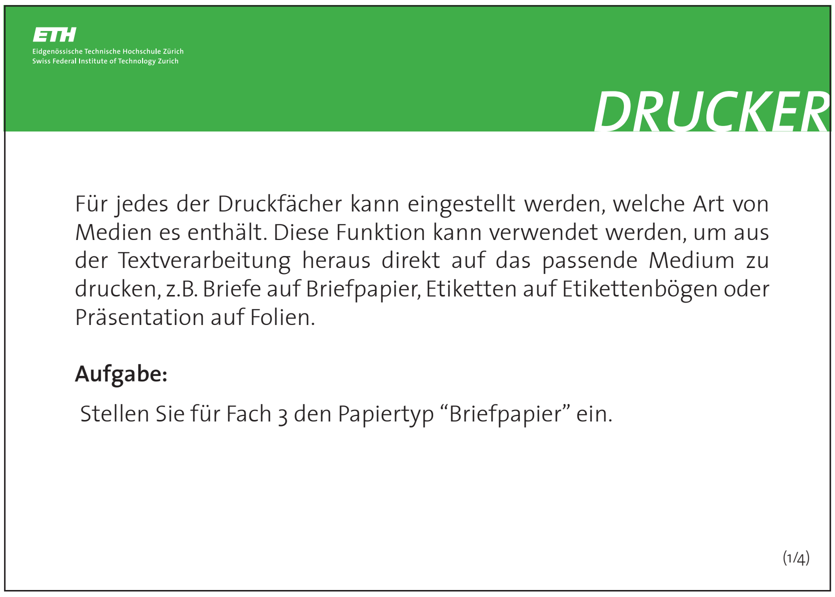


RADIO

Aufgabe:

Hören Sie DRS 3.

(3/3)



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

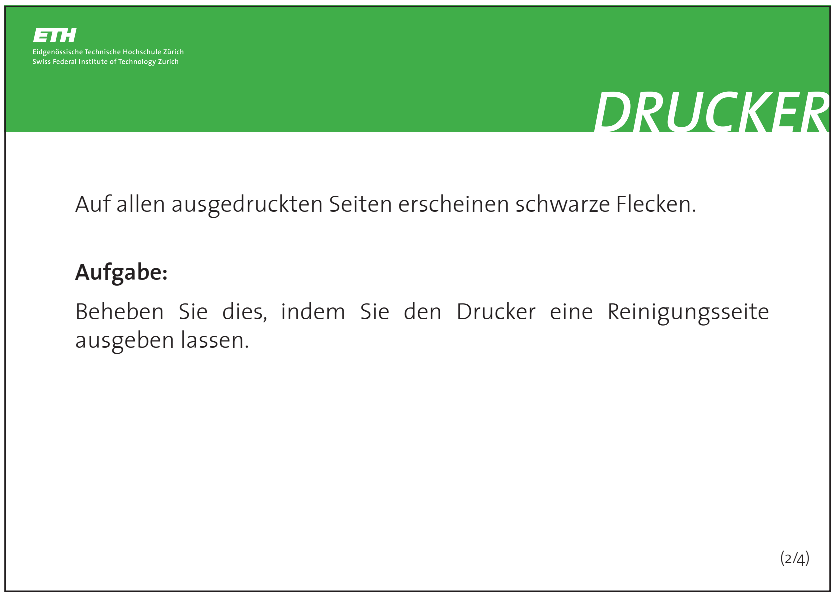
DRUCKER

Für jedes der Druckfächer kann eingestellt werden, welche Art von Medien es enthält. Diese Funktion kann verwendet werden, um aus der Textverarbeitung heraus direkt auf das passende Medium zu drucken, z.B. Briefe auf Briefpapier, Etiketten auf Etikettenbögen oder Präsentation auf Folien.

Aufgabe:

Stellen Sie für Fach 3 den Papiertyp "Briefpapier" ein.

(1/4)



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

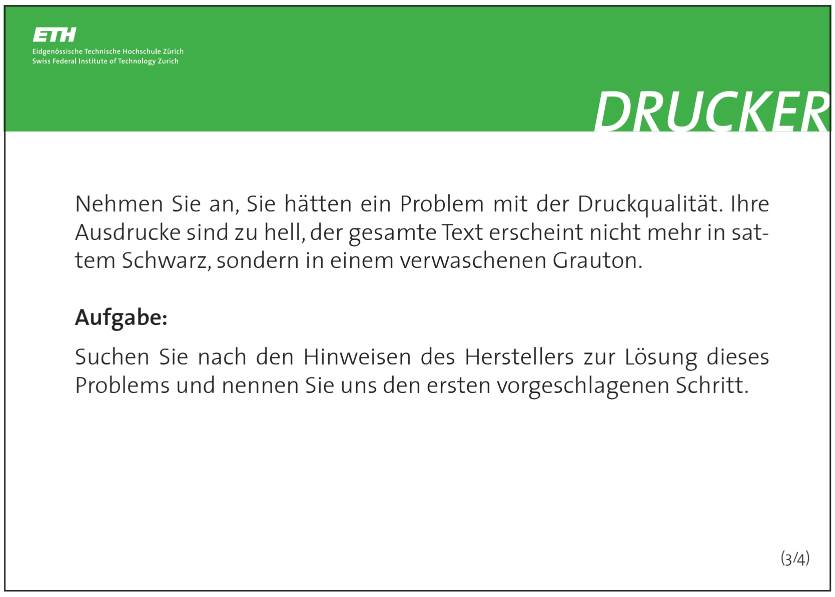
DRUCKER

Auf allen ausgedruckten Seiten erscheinen schwarze Flecken.

Aufgabe:

Beheben Sie dies, indem Sie den Drucker eine Reinigungsseite ausgeben lassen.

(2/4)



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

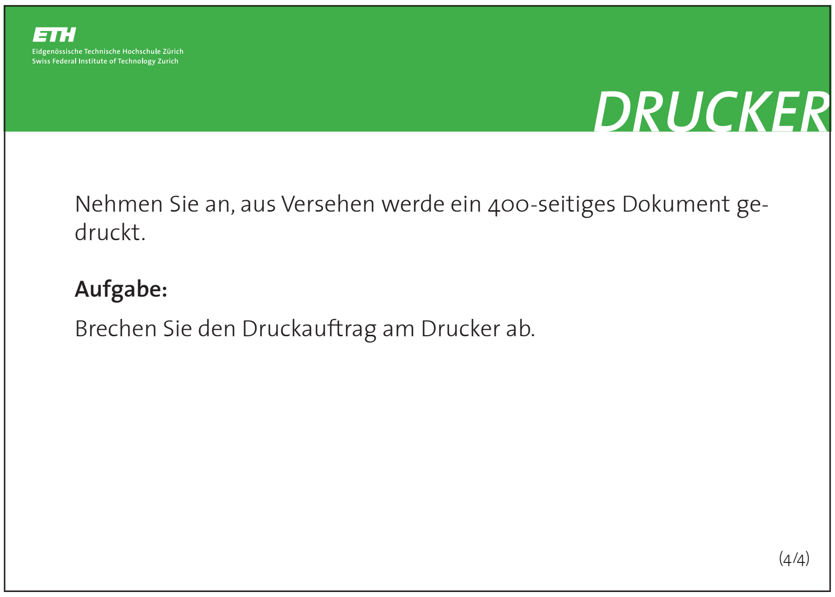
DRUCKER

Nehmen Sie an, Sie hätten ein Problem mit der Druckqualität. Ihre Ausdrucke sind zu hell, der gesamte Text erscheint nicht mehr in sattem Schwarz, sondern in einem verwaschenen Grauton.

Aufgabe:

Suchen Sie nach den Hinweisen des Herstellers zur Lösung dieses Problems und nennen Sie uns den ersten vorgeschlagenen Schritt.

(3/4)



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

DRUCKER

Nehmen Sie an, aus Versehen werde ein 400-seitiges Dokument gedruckt.

Aufgabe:

Brechen Sie den Druckauftrag am Drucker ab.

(4/4)

B. BIT API

This appendix describes the BIT API, which was briefly outlined in Section 4.7.9. For every API function, the following reference indicates the applet type, mode, and state in which it can be called.

General

- `bit.out(object)`

  Inserts a Lua object's string representation in a BITML view. This function cannot be used in a meaningful way outside of a processing instruction.

  Arguments: `object`: object to be inserted

  Return value: —

  Applet type: permanent, transitory

  Applet mode: exclusive

  Applet state: coupled, uncoupled

- `bit.debug(object)`

  Prints a Lua object's string representation to the console. This function can be used when debugging an applet.

  Arguments: `object`: object to be printed

  Return value: —

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

User Interface

- `bit.show_view(view_name, args)`

Displays a given BITML view on the screen. An associative array can be passed to the view, which will be available during rendering in the `args` variable.

Arguments: `view_name`: The name of the view to be displayed. The view will be loaded from a file named `views/view_name.bit`. `args`: An optional associative arrray that will be made available to scripts that are executed as the view is rendered.

Return value: —

Applet type: permanent, transitory

Applet mode: exclusive

Applet state: coupled, uncoupled

- `bit.set_aggregation_text(aggr_text)`

  This function allows applets running in aggregation mode to specify textual output that will be displayed by the browser. The browser will display the text on a best effort basis. However, an implementation is free to truncate the output as needed.

  Arguments: `aggr_text`: string containing text to be displayed

  Return value: —

  Applet type: permanent

  Applet mode: aggregation

  Applet state: coupled

- `bit.set_aggregation_image(aggr_img)`

  This function allows applets running in aggregation mode to specify an icon image that will be displayed by the browser. Browsers can scale images to fit the space that is allocated for an applet's output.

  Arguments: `aggr_img`: A string referencing the image to be displayed. The string can either be a URL, referencing an image on the internet, or the name of a local file.

  Return value: —

  Applet type: permanent

  Applet mode: aggregation

  Applet state: coupled

- `bit.show_note(text, type)`

  Displays a modal textual pop-up note.

  Arguments: `text`: A string containing the text to be displayed. `type`: A string that specifies the nature of the note. Valid types are “info”, “error”, or “confirmation”.

  Return value: —

  Applet type: permanent, transitory

  Applet mode: exclusive

  Applet state: coupled, uncoupled

- `bit.show_dialog(text, type)`

  Displays a modal dialog with a single input field to collect data entered by the user.

  Arguments: `text`: A string containing the text to be displayed in the dialog. `type`: A string that specifies the type of input to be collected. Valid types are “text”, “integer”, “date”, “time”, “tag_id” (for manually entering an object’s tag ID), “confirmation” (for confirming or canceling an operation).

  Return value: Data entered by the user. The value returned is independent of the current locale, i.e., user input is converted into a standard format.

  Applet type: permanent, transitory

  Applet mode: exclusive

  Applet state: coupled, uncoupled

- `bit.open_in_web_browser(url)`

  Opens a URL in the device’s standard web browser.

  Arguments: `url`: URL to be opened

  Return value: —

  Applet type: permanent, transitory

  Applet mode: exclusive

  Applet state: coupled, uncoupled

- `bit.vibrate(duration)`

Switches on the device's vibration module for the specified duration.

Arguments: `duration`: time in milliseconds

Return value: —

Applet type: permanent, transitory

Applet mode: exclusive

Applet state: coupled, uncoupled

Persistent Storage

- `bit.storage_put(key, object)`

  Serializes a Lua object and persistently stores it under a given key. Existing entries stored under the same key will be overwritten. Note that every applet has its own key namespace. Two different applets cannot read/write an object stored under the same key.

  Arguments: `key`: An arbitrary string identifying the object. `object`: The object to be stored.

  Return value: `true` if the operation succeeded, `false` otherwise.

  Applet type: permanent

  Applet mode: exclusive

  Applet state: coupled, uncoupled

- `bit.storage_get(key)`

  Retrieves and deserializes a Lua object that has been stored persistently under the given key. Note that every applet has its own key namespace. Two different applets cannot read/write an object stored under the same key.

  Arguments: `key`: string identifying the object to be retrieved

  Return value: previously stored Lua object, `nil` if no object was found

  Applet type: permanent

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

- `bit.storage_remove(key)`

  Deletes a Lua object that has been stored persistently under the given key. Note that every applet has its own key namespace. Two different applets cannot read/write an object stored under the same key.

  Arguments: `key`: string identifying the object to be deleted from the persistent storage

  Return value: `true` if success, `false` otherwise (object not found, I/O error, etc.)

  Applet type: permanent

  Applet mode: exclusive

  Applet state: coupled, uncoupled

- `bit.storage_has(key)`

  Checks whether an object has been stored under the given key. Note that every applet has its own key namespace. Two different applets cannot read/write an object stored under the same key.

  Arguments: `key`: string identifying the object

  Return value: `true` if object exists, `false` otherwise

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

Data Exchange

- `bit.http_request(url, method, data, headers)`

  Sends an HTTP request to a server.

  Arguments: `url`: The server's URL. `method`: The HTTP method (e.g., "GET", "POST", etc.). `data`: The data to send along with the request (for "POST" and "PUT" requests). `headers`: List of additional HTTP headers to include in the request.

  Return value: numeric HTTP status code or `nil` in case of client-side error

  Applet type: permanent, transitory

Applet mode: exclusive

Applet state: coupled, uncoupled

- `bit.json_decode(json)`

  Deserializes a Lua object from its JSON representation. JSON data is mapped into Lua tables according to the following rules: 1. JSON objects are converted into Lua dictionary tables. 2. JSON arrays are converted into numerically indexed Lua array tables. 3. All other JSON data types are converted into their corresponding Lua types.

  Arguments: `json`: string in JSON format to be deserialized

  Return value: deserialized Lua object

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

- `bit.json_encode(object)`

  Serializes a Lua object in JSON format. See `bit.json_decode(json)` for mapping rules.

  Arguments: `object`: Lua object to be serialized

  Return value: string containing the JSON representation of the object

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

- `bit.xml_decode(xml)`

  Deserializes a Lua object from its XML representation. XML trees are mapped into Lua tables according to the following rules:[1] 1. For every XML element, a separate Lua table is created. 2. Subelements can be accessed through numerical indices. 3. Element names and attributes can be accessed through string keys.

  Arguments: `xml`: string in JSON format to be deserialized

  Return value: deserialized Lua object

[1]This mapping is inspired by the LuaXML module. See `www.viremo.de/LuaXML` for details on use.

Applet type: permanent, transitory

Applet mode: aggregation, exclusive

Applet state: coupled, uncoupled

- `bit.xml_encode(object)`

  Serializes a Lua object in XML format.

  Arguments: `object`: Lua object to be serialized

  Return value: string containing the XML representation of the object

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

Open Lookup Infrastructure

- `bit.get_repository(url)`

  Returns a `Repository` proxy object representing the resource repository running at a given URL. The proxy object can be used to invoke operations on the remote resource repository (see below).

  Arguments: `url`: resource repository's URL

  Return value: `Repository` proxy object

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

- `bit.get_manufacturer_repository(tagged_object)`

  Return's a `Repository` proxy object representing the resource repository run by the manufacturer of a given tagged object.

  Arguments: `tagged_object`: tagged object whose manufacturer resource repository is to be returned

  Return value: `Repository` proxy object

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

- `Repository.get_resource_description(resource_id)`

  Retrieves a resource description with a given ID from the resource repository.

  Arguments: `resource_id`: resource description's ID

  Return value: resource description

  Applet type: permanent, transitory

  Applet mode: exclusive

  Applet state: coupled, uncoupled

- `Repository.lookup_resource(criteria)`

  Retrieves the descriptions of all resources matching the given criteria from a resource repository.

  Arguments: `criteria`: Lua table (used as an associative array) to specify query parameters. The table can contain the following keys: "tag_id", "profile", "context", "max_results", "start_index" (see Section 3.4.2 for details on query parameters). In aggregation mode, the lookup criteria are subject to restrictions as discussed in Section 4.8.3.

  Return value: list of resource descriptions

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

- `bit.get_search_service(url)`

  Returns a `SearchService` proxy object representing the search service running at a given URL. The proxy object can be used to invoke operations on the remote search service (see below).

  Arguments: `url`: search service's URL

  Return value: `SearchService` proxy object

  Applet type: permanent, transitory

  Applet mode: aggregation, exclusive

  Applet state: coupled, uncoupled

- `SearchService.search(criteria)`

Retrieves the descriptions of all resources matching the given criteria from a search service.

Arguments: `criteria`: Lua table (used as an associative array) to specify search parameters. The table can contain the following keys: "tag_id", "profile", "context", "max_results", "start_index" (see Section 3.4.3 for details). In aggregation mode, the search criteria are subject to restrictions as discussed in Section 4.8.3.

Return value: list of resource descriptions

Applet type: permanent, transitory

Applet mode: aggregation, exclusive

Applet state: coupled, uncoupled

Bibliography

[1] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, October 1997.

[2] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. UIML: an appliance-independent XML user interface language. *Computer Networks*, 31(11–16):1695–1708, 1999.

[3] Robert Adelmann, Marc Langheinrich, and Christian Floerkemeier. Toolkit for Bar Code Recognition and Resolving on Camera Phones – Jump Starting the Internet of Things. In *Informatik 2006 Workshop on Mobile and Embedded Interactive Systems (MEIS'06)*, Dresden, Germany, October 2006.

[4] Heikki Ailisto, Lauri Pohjanheimo, Pasi Välkkynen, Esko Strömmer, Timo Tuomisto, and Ilkka Korhonen. Bridging the physical and virtual worlds by local connectivity-based physical selection. *Personal Ubiquitous Computing*, 10(6):333–344, 2006.

[5] Glen Allmendinger and Ralph Lombreglia. Four Strategies for the Age of Smart Services. *Harvard Business Review*, 83(10):131–145, October 2005.

[6] Abhaya Asthana, Mark Cvavatts, and Paul Krzyzanowski. An Indoor Wireless System for Personalized Shopping Assistance. In *Proceedings of the First Workshop on Mobile Computing Systems and Applications (WMCSA 1994)*, pages 69–74, Santa Cruz, CA, USA, December 1994.

[7] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In *Proceedings of the First International Conference on Pervasive Computing*, volume 2414 of *Lecture*

Notes in Computer Science, pages 195–210, Zurich, August 2002. Springer.

[8] Rafael Ballagas, Jan Borchers, Michael Rohs, and Jennifer G. Sheridan. The Smart Phone: A Ubiquitous Input Device. *IEEE Pervasive Computing*, 5(1):70–77, 2006.

[9] Rafael Ballagas, Michael Rohs, Jennifer G. Sheridan, and Jan Borchers. BYOD: Bring Your Own Device. In *UbiComp 2004 Workshop on Ubiquitous Display Environments*, Nottingham, UK, 2004.

[10] Michael Beigl. Point & Click – Interaction in Smart Environments. In *HUC'99: First International Symposium on Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 311–313, Karlsruhe, Germany, September 1999. Springer.

[11] Rachel Bellamy, Calvin Swart, Wendy A. Kellogg, John Richards, and Jonathan Brezin. Designing an E-Grocery Application for a Palm Computer: Usability and Interface Issues. *IEEE Personal Communications*, 8(4):60–64, August 2001.

[12] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. W3C Candidate Recommendation, September 2009. `www.w3.org/TR/2009/CR-CSS2-20090908`.

[13] John M. Boyer. XForms 1.0 (Third Edition). W3C Recommendation, October 2007. `www.w3.org/TR/xforms/`.

[14] Adam B. Brody and Edward J. Gottsman. Pocket BargainFinder: A Handheld Device for Augmented Commerce. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 44–51, Karlsruhe, Germany, September 1999. Springer.

[15] Gregor Broll, Markus Haarländer, Massimo Paolucci, Matthias Wagner, Enrico Rukzio, and Albrecht Schmidt. Collect&Drop: A Technique for Multi-Tag Interaction with Real World Objects and Information. In *Proceedings of the European Conference on Ambient Intelligence (AmI 2007)*, volume 5355 of *Lec-*

ture Notes in Computer Science, pages 175–191, Nuremberg, Germany, November 2008. Springer.

[16] Gregor Broll, Sven Siorpaes, Enrico Rukzio, Massimo Paolucci, John Hamard, Matthias Wagner, and Albrecht Schmidt. Supporting Mobile Service Usage through Physical Mobile Interaction. In *Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications (Percom 2007)*, pages 262–271, White Plains, NY, USA, March 2007. IEEE Computer Society.

[17] A.J. Bernheim Brush, Tammara Combs Turner, Marc A. Smith, and Neeti Gupta. Scanning Objects in the Wild: Assessing an Object Triggered Information System. In *Proceedings of the 70th International Conference on Ubiquitous Computing (UbiComp 2005)*, volume 3660 of *Lecture Notes in Computer Science*, pages 305–322, Tokyo, Japan, September 2005. Springer.

[18] Marc Bühler. Universal Browser and Interaction Service. Master's thesis, Institute for Pervasive Computing, Department of Computer Science, ETH Zurich, Zurich, Switzerland, 2007.

[19] Scott Carter, Elizabeth Churchill, Laurent Denoue, Jonathan Helfman, and Les Nelson. Digital Graffiti: Public Annotation of Multimedia Content. In *CHI '04: CHI '04 Extended Abstracts on Human Factors in Computing Systems*, pages 1207–1210, Vienna, Austria, 2004.

[20] Gabriella Castelli, Alberto Rosi, Marco Mamei, and Franco Zambonelli. A Simple Model and Infrastructure for Context-aware Browsing of the World. In *Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications (Percom 2007)*, pages 229–238, White Plains, NY, USA, March 2007. IEEE Computer Society.

[21] David M. Chess. Security Issues in Mobile Code Systems. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 1–14. Springer, Berlin Heidelberg New York, 1998.

[22] Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a Context-aware Electronic

Tourist Guide: Some Issues and Experiences. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2000)*, pages 17–24, The Hague, The Netherlands, April 2000. ACM.

[23] Luca Chittaro and Daniele Nadalutti. Presenting Evacuation Instructions on Mobile Devices by means of Location-Aware 3D Virtual Environments. In *Proceedings of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI 2008)*, pages 395–398, Amsterdam, The Netherlands, September 2008. ACM. Short paper.

[24] Douglas Crockford. RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON), July 2006.

[25] Ali Dada, Thorsten Staake, and Felix von Reischach. Displaying Dynamic Carbon Footprints of Products on Mobile Phones (Demo). In *Advances in Pervasive Computing – Adjunct Proceedings of the 6th International Conference on Pervasive Computing (Pervasive 2008)*, Sydney, Australia, May 2008. OCG.

[26] Nigel Davies and Hans-Werner Gellersen. Beyond Prototypes: Challenges in Deploying Ubiquitous Systems. *IEEE Pervasive Computing*, 1(1):26–35, 2002.

[27] Philippe Debaty and Deborah Caswell. Uniform Web Presence Architecture for People, Places, and Things. *IEEE Personal Communications*, 8(4):46–51, August 2001.

[28] Lisa Deng and Landon P. Cox. LiveCompare: Grocery Bargain Hunting Through Participatory Sensing. In *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications (HotMobile 2009)*, Santa Cruz, CA, USA, February 2009. ACM.

[29] Anind K. Dey. Understanding and Using Context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

[30] W. Keith Edwards, Mark W. Newman, Jana Sedivy, Trevor Smith, and Shahram Izadi. Challenge: Recombinant Computing and the Speakeasy Approach. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM 2002)*, pages 279–286, Atlanta, GA, USA, September 2002. ACM.

[31] EPCglobal. EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz, Version 1.1.0, December 2005. www.epcglobalinc.org/standards/uhfc1g2/uhfc1g2_1_1_0-standard-20071017.pdf.

[32] EPCglobal. EPC Information Services (EPCIS) Version 1.0.1 Specification, September 2007. www.epcglobalinc.org/standards/epcis/epcis_1_0_1-standard-20070921.pdf.

[33] EPCglobal. EPCglobal Object Name Service (ONS) 1.0.1 – Ratified Standard Specification with Approved, Fixed Errata, May 2008. www.epcglobalinc.org/standards/ons/ons_1_0_1-standard-20080529.pdf.

[34] EPCglobal. EPCglobal Tag Data Standards Version 1.4, June 2008. www.epcglobalinc.org/standards/tds/tds_1_4-standard-20080611.pdf.

[35] EPCglobal. The EPCglobal Architecture Framework – EPCglobal Final Version 1.3, March 2009. www.epcglobalinc.org/standards/architecture/architecture_1_3-framework-20090319.pdf.

[36] Ulrich Etter. BIT – A Browser for the Internet of Things. Master's thesis, Institute for Pervasive Computing, Department of Computer Science, ETH Zurich, Zurich, Switzerland, 2009.

[37] Kevin F. Eustice, Tobin J. Lehman, Armando Morales, Michelle C. Munson, Stefan Edlund, and Miguel Guillen. A universal information appliance. *IBM Systems Journal*, 38(4):575–601, 1999.

[38] Andrew Fano and Anatole Gershman. The Future of Business Services in the Age of Ubiquitous Computing. *Communications of the ACM*, 45(12):83–87, 2002.

[39] Andrew E. Fano. Shopper's Eye: Using Location-based Filtering for a Shopping Agent in the Physical World. In *Proceedings of the 2nd International Conference on Autonomous Agents*, pages 416–421, Mineapolis, MN, USA, 1998. ACM.

[40] Glover T. Ferguson. Have Your Objects Call My Objects. *Harvard Business Review*, 80(6):138–144, June 2002.

[41] Roy T. Fielding and Richard N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.

[42] Elgar Fleisch and Christian Tellkamp. Inventory inaccuracy and supply chain performance: a simulation study of a retail supply chain. *International Journal of Production Economics*, 95(3):373–385, 2005.

[43] B. J. Fogg. Persuasive Computers: Perspectives and Research Directions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1998)*, pages 225–232, Los Angeles, CA, USA, April 1998. ACM / Addison-Wesley Publishing Co.

[44] InterNational Committee for Information Technology Standards. V2 – Information Technology Access Interfaces, August 2005. `http://v2.incits.org/`.

[45] Christian Frank, Philipp Bolliger, Christof Roduner, and Wolfgang Kellerer. Objects Calling Home: Locating Objects Using Mobile Phones. In *Proceedings of the 5th International Conference on Pervasive Computing (Pervasive 2007)*, volume 4480 of *Lecture Notes in Computer Science*, pages 351–368, Toronto, Canada, May 2007. Springer.

[46] Adrian Friday, Nigel Davies, Nat Wallbank, Elaine Catterall, and Stephen Pink. Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments. *Wireless Networks*, 10(6):631–641, November 2004.

[47] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[48] Krzysztof Gajos and Daniel S. Weld. SUPPLE: Automatically Generating User Interfaces. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, pages 93–100, New York, NY, USA, 2004. ACM Press.

[49] Alan Ganguillet. Universal Appliance Helper. Diploma thesis, Department of Design, Zurich University of the Arts (ZHdK), Zurich, Switzerland, 2006.

[50] Jesse J. Garrett. Ajax: A New Approach to Web Applications, February 2005. www.adaptivepath.com/ideas/essays/archives/000385.php.

[51] Christian Giordano, Sonia Modeo, Giorgio Bernardi, and Ferdinando Ricchiuti. Using Mobile Phones as Remote Control for Ubiquitous Video-Recording. In *Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia (MUM 2007)*, pages 125–130, Oulu, Finland, December 2007. ACM.

[52] Ben Goodger, Ian Hickson, David Hyatt, and Chris Waterson. XML User Interface Language (XUL) 1.0, 2001. www.mozilla.org/projects/xul/xul.html.

[53] Chris Greenhalgh, Steve Benford, Adam Drozd, Martin Flintham, Alastair Hampshire, Leif Oppermann, Keir Smith, and Christoph von Tycowicz. Addressing Mobile Phone Diversity in Ubicomp Experience Development. In *UbiComp '07: Proceedings of the 9th International Conference on Ubiquitous Computing*, volume 4717 of *Lecture Notes in Computer Science*, pages 447–464, Innsbruck, Austria, September 2007. Springer.

[54] William G. Griswold, Patricia Shanahan, Steven W. Brown, Robert Boyer, Matt Ratto, R. Benjamin Shapiro, and Tan Minh Truong. ActiveCampus: Experiments in Community-Oriented Ubiquitous Computing. *Computer*, 37(10):73–81, 2004.

[55] GS1 US. An Introduction to the Global Trade Item Nnumber (GTIN), December 2006. http://barcodes.gs1us.org/dnn_bcec/Documents/tabid/136/DMXModule/731/Command/Core_Download/Default.aspx?EntryId=59.

[56] Erik Guttman. Service Location Protocol: Automatic Discovery of IP Network Services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[57] Gerald Häubl and Valerie Trifts. Consumer Decision Making in Online Shopping Environments: The Effects of Interactive Decision Aids. *Marketing Science*, 19(1):4–21, 2000.

[58] Sumi Helal, Nitin Desai, Varun Verma, and Choonhwa Lee. Konark - A Service Discovery and Delivery Protocol for Ad-Hoc Networks. In *IEEE Wireless Communications and Networking*

Conference (WCNC 2003), volume 3, pages 2107–2113, March 2003.

[59] Christopher K. Hess and Roy H. Campbell. A Context File System for Ubiquitous Computing Environments. Technical Report UIUCDCS-R-2002-2285 UILU-ENG-2002-1729, University of Illinois at Urbana-Champaign, July 2002.

[60] Todd D. Hodes, Randy H. Katz, Edouard Servan-Schreiber, and Lawrence Rowe. Composable Ad-hoc Mobile Services for Universal Interaction. In *MobiCom '97: Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 1–12, Budapest, Hungary, September 1997. ACM Press.

[61] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes. The Evolution of Lua. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (HOPL III)*, pages 2.1–2.26, San Diego, CA, USA, June 2007.

[62] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. Lua – An Extensible Extension Language. *Software: Practice and Experience*, 26(6):635–652, June 1996.

[63] Stephen S. Intille. Designing a Home of the Future. *IEEE Pervasive Computing*, 1(2):76–82, 2002.

[64] Charles L. Isbell, Olufisayo Omojokun, and Jeffrey S. Pierce. From devices to tasks: automatic task prediction for personalized appliance control. *Personal Ubiquitous Computing*, 8(3-4):146–153, 2004.

[65] Hiroshi Ishii and Brygg Ullmer. Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms. In *CHI '97: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 234–241, Atlanta, GA, USA, March 1997. ACM Press.

[66] Jerry Kang and Dana Cuff. Pervasive Computing: Embedding the Public Sphere. *Washington and Lee Law Review*, 62:93–146, 2005.

[67] Khomkrit Kaowthumrong, John Lebsack, and Richard Han. Automated Selection of the Active Device in Interactive Multi-Device Smart Spaces. In *Proceedings of the Ubicomp 2002 Workshop on Supporting Spontaneous Interaction in Ubiquitous Computing Settings*, Göteborg, Sweden, September 2002.

[68] Alan H. Karp. Lessons from E-speak. In *Proceedings of the 1st Workshop on Real, Large, Distributed Systems (WORLDS 2004)*, San Francisco, CA, USA, December 2004. USENIX. `www.usenix.org/events/worlds04/tech/full_papers/karp/karp.pdf`.

[69] Wooyoung Kim, Sven Graupner, Akhil Sahai, Dmitry Lenkov, Chetan Chudasama, Samuel Whedbee, Yuhua Luo, Bharati Desai, Howard Mullings, and Pui Wong. Web E-Speak: Facilitating Web-Based E-Services. *IEEE MultiMedia*, 9(1):43–55, 2002.

[70] Tim Kindberg. Implementing Physical Hyperlinks Using Ubiquitous Identifier Resolution. In *Proceedings of the 11th International Conference on World Wide Web (WWW 2002)*, pages 191–199, Honolulu, HI, USA, May 2002. ACM.

[71] Tim Kindberg and John Barton. A Web-based nomadic computing system. *Computer Networks*, 35(4):443–456, March 2001.

[72] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, Places, Things: Web Presence for the Real World. *Mobile Networks and Applications*, 7(5):365–376, 2002.

[73] Tiiu Koskela and Kaisa Väänänen-Vainio-Mattila. Evolution towards smart home environments: empirical evaluation of three user interfaces. *Personal Ubiquitous Computing*, 8(3-4):234–240, 2004.

[74] Panos Kourouthanassis and George Roussos. Developing Consumer-Friendly Pervasive Retail Systems. *IEEE Pervasive Computing*, 2(2):32–39, April 2003.

[75] Hannu Kukka, Timo Ojala, Juha Tiensyrjä, and Teemu Mikkonen. panOULU Luotsi: A Location Based Information Mash-up with XML Aggregator and WiFi Positioning. In *Proceedings of*

the 7th International Conference on Mobile and Ubiquitous Multimedia (MUM 2008), pages 80–83, Umeå, Sweden, December 2008.

[76] Matthias Lampe, Christian Metzger, Elgar Fleisch, and Oliver Zweifel. Digitally Augmented Collectibles. Adjunct Proceedings of 8th Annual ACM Symposium on User Interface Software and Technology (UIST), Seattle, October 2005.

[77] Bill LaPlant, Shari Trewin, Gottfried Zimmermann, and Gregg Vanderheiden. The Universal Remote Console: A Universal Access Bus for Pervasive Computing. *IEEE Pervasive Computing*, 3(1):76–80, 2004.

[78] R.D. Lawrence, G.S. Almasi, V. Kotlyar, M.S. Viveros, and S.S. Duri. Personalization of Supermarket Product Recommendations. *Data Mining and Knowledge Discovery*, 5(1–2):11–32, January 2001.

[79] Lawrence Lessig. *The Future of Ideas: The Fate of the Commons in a Connected World*. Random House Inc., New York, NY, USA, 2001.

[80] Laura M. Leventhal and Julie A. Barnes. *Usability Engineering: Process, Products, and Examples*. Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2008.

[81] Theodore Levitt. Marketing intangible products and product intangibles. *Harvard Business Review*, 59(3):94–102, May 1981.

[82] Clayton Lewis and John Rieman. *Task-Centered User Interface Design – A Practical Introduction*. University of Colorado, 1993.

[83] Henry Lieberman and José Espinosa. A Goal-Oriented Interface to Consumer Electronics Using Planning and Commonsense Reasoning. In *IUI '06: Proceedings of the 11th International Conference on Intelligent User Interfaces*, pages 226–233, New York, NY, USA, 2006. ACM Press.

[84] Peter Ljungstrand, Johan Redström, and Lars Erik Holmquist. WebStickers: Using Physical Tokens to Access, Manage and Share Bookmarks to the Web. In *DARE '00: Proceedings of DARE 2000 on Designing Augmented Reality Environments*, pages 23–31, Elsinore, Denmark, April 2000. ACM Press.

[85] Andreas Lorenz, Clara Fernandez De Castro, and Enrico Rukzio. Using Handheld Devices for Mobile Interaction with Displays in Home Environments. In *Proceedings of the 11th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI 2009)*, Bonn, Germany, September 2009.

[86] S. McFaddin, D. Coffman, J.H. Han, H.K. Jang, J.H. Kim, J.K. Lee, M.C. Lee, Y.S. Moon, C. Narayanaswami, Y.S. Paik, J.W. Park, and D. Soroker. Modeling and Managing Mobile Commerce Spaces Using RESTful Data Services. In *Proceedings of the 9th International Conference on Mobile Data Management (MDM 2008)*, pages 81–89, Beijing, China, April 2008.

[87] Michael Mealling. RFC 3403: Dynamic Delegation Discovery System (DDDS) – Part Three: The Domain Name System (DNS) Database, October 2002.

[88] Alan Messer, Anugeetha Kunjithapatham, Mithun Sheshagiri, Henry Song, Praveen Kumar, Phuong Nguyen, and Kyoung Hoon Yi. InterPlay: A Middleware for Seamless Device Integration and Task Orchestration in a Networked Home. In *Proceedings of the Fourth IEEE International Conference on Pervasive Computing and Communications (PerCom'06)*, pages 296–307, Pisa, Italy, March 2006. IEEE Computer Society.

[89] Martin Peter Michael and Mohsen Darianian. Architectural Solutions for Mobile RFID Services for the Internet of Things. In *Proceedings of IEEE Congress on Services (SERVICES 2008)*, pages 71–74, Honolulu, HI, USA, July 2008. IEEE Computer Society.

[90] Microsoft. Extensible Application Markup Language (XAML), June 2008. `http://go.microsoft.com/fwlink/?LinkId=113699`.

[91] Nikola Mitrovic and Eduardo Mena. Adaptive User Interface for Mobile Devices. In *Proceedings of the 9th International Workshop on the Design, Specification, and Verification of Interactive Systems (DSV-IS 2002)*, volume 2545 of *Lecture Notes in Computer Science*, pages 29–43, Rostock, Germany, June 2002. Springer.

[92] Giulio Mori, Fabio Paternò, and Carmen Santoro. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Transactions on Software Engineering*, 28(8):797–813, August 2002.

[93] Giulio Mori, Fabio Paterno, and Carmen Santoro. Tool Support for Designing Nomadic Applications. In *Proceedings of the 8th International Conference on Intelligent User Interfaces (IUI 2003)*, pages 141–148, Miami, FL, USA, January 2003.

[94] Giulio Mori, Fabio Paterno, and Carmen Santoro. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Transactions on Software Engineering*, 30(8):507–520, August 2004.

[95] Andreas Müller, Peter Forbrig, and Clemens H. Cap. Model-Based User Interface Design Using Markup Concepts. In *Proceedings of the 8th International Workshop on the Design, Specification, and Verification of Interactive Systems (DSV-IS 2001)*, volume 2220 of *Lecture Notes in Computer Science*, pages 16–27, Glasgow, Scotland, UK, June 2001. Springer.

[96] Ilario Musio. Interacting with Appliances using Mobile Phones. Master's thesis, Institute for Pervasive Computing, Department of Computer Science, ETH Zurich, Zurich, Switzerland, 2006.

[97] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):3–28, March 2000.

[98] Brad A. Myers, Jeffrey Nichols, Jacob O. Wobbrock, and Robert C. Miller. Taking Handheld Devices to the Next Level. *IEEE Computer*, 37(12):36–43, 2004.

[99] Brad A. Myers, Herb Stiel, and Robert Gargiulo. Collaboration using multiple PDAs connected to a PC. In *CSCW '98: Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pages 285–294, Seattle, WA, USA, 1998.

[100] Erica Newcomb, Toni Pashley, and John Stasko. Mobile Computing in the Retail Arena. In *Proceedings of the SIGCHI Con-*

ference on Human Factors in Computing Systems (CHI 2003), pages 337–344, Ft. Lauderdale, FL, USA, April 2003. ACM.

[101] Mark W. Newman, Ame Elliott, and Trevor F. Smith. Providing an Integrated User Experience of Networked Media, Devices, and Services through End-User Composition. In *Proceedings of the 6th International Conference on Pervasive Computing (Pervasive 2008)*, volume 5013 of *Lecture Notes in Computer Science*, pages 213–227, Sydney, Australia, May 2008. Springer.

[102] NFC Forum. Near Field Communication (NFC). `www.nfc-forum.org`.

[103] NFC Forum. Connection Handover 1.1 – Technical Specification, November 2008. `www.nfc-forum.org/specs/`.

[104] Jeffrey Nichols and Brad A. Myers. Studying the Use of Handhelds to Control Smart Appliances. In *ICDCSW '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 274–279, Washington, DC, USA, May 2003. IEEE Computer Society.

[105] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. Generating Remote Control Interfaces for Complex Appliances. In *UIST '02: Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology*, pages 161–170, Paris, France, October 2002. ACM Press.

[106] Jeffrey Nichols, Brad A. Myers, and Brandon Rothrock. UNIFORM: Automatically Generating Consistent Remote Control User Interfaces. In *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 611–620, New York, NY, USA, 2006. ACM Press.

[107] Jeffrey Nichols, Brandon Rothrock, Duen Horng Chau, and Brad A. Myers. Huddle: Automatically Generating Interfaces for Systems of Multiple Connected Appliances. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST 2006)*, pages 279–288, Montreux, Switzerland, October 2006.

[108] Don Norman. *The Design of Everyday Things.* Basic Books, New York, NY, USA, 2002.

[109] Stina Nylander, Markus Bylund, and Annika Waern. Ubiquitous service access through adapted user interfaces on multiple devices. *Personal Ubiquitous Computing*, 9(3):123–133, 2005.

[110] Dan R. Olsen, Sean Jefferies, Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-modal Interaction using XWeb. In *UIST '00: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 191–200, San Diego, CA, USA, November 2000. ACM Press.

[111] Olufisayo Omojokun, Jeffrey S. Pierce, Charles L. Isbell, and Prasun Dewan. Comparing end-user and intelligent remote control interface generation. *Personal Ubiquitous Computing*, 10(2):136–143, 2006.

[112] Jason Pascoe, Nick Ryan, and David Morse. Using While Moving: HCI Issues in Fieldwork Environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(3):417–437, September 2000.

[113] Lauri Pohjanheimo, Heikki Keränen, and Heikki Ailisto. Implementing TouchMe Paradigm with a Mobile Phone. In *sOc-EUSAI '05: Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence*, pages 87–92, Grenoble, France, October 2005. ACM Press.

[114] Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *UbiComp '01: Proceedings of the 3rd International Conference on Ubiquitous Computing*, volume 2201 of *Lecture Notes in Computer Science*, pages 56–75, Atlanta, GA, USA, 2001. Springer.

[115] Angel Puerta and Jacob Eisenstein. XIML: A Common Representation for Interaction Data. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI 2002)*, pages 214–215, San Francisco, CA, USA, January 2002. ACM.

[116] Jun Rekimoto and Katashi Nagao. The World through the Computer: Computer Augmented Interaction with Real World En-

vironments. In *UIST '95: Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, pages 29–36, Pittsburgh, PA, USA, November 1995. ACM Press.

[117] Jan S. Rellermeyer, Gustavo Alonso, and Timothy Roscoe. R-OSGi: Distributed Applications Through Software Modularization. In *Proceedings of the 8th ACM/IFIP/USENIX International Conference on Middleware (Middleware 2007)*, volume 4834 of *Lecture Notes in Computer Science*, pages 1–20, Newport Beach, CA, USA, November 2007. Springer.

[118] Jan S. Rellermeyer, Oriana Riva, and Gustavo Alonso. AlfredO: An Architecture for Flexible Interaction with Electronic Devices. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware 2008)*, volume 5346 of *Lecture Notes in Computer Science*, pages 22–41, Leuven, Belgium, December 2008. Springer.

[119] Florian Resatsch, Stephan Karpischek, Uwe Sandner, and Stephan Hamacher. Mobile Sales Assistant – NFC for Retailers. In *Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI 2007)*, pages 313–316, Singapore, September 2007. ACM.

[120] Jukka Riekki, Ivan Sanchez, and Mikko Pyykkönen. Universal Remote Control for the Smart World. In *Proceedings of the 5th International Conference on Ubiquitous Intelligence and Computing (UIC 2008)*, volume 5061 of *Lecture Notes in Computer Science*, pages 563–577, Oslo, Norway, June 2008. Springer.

[121] Jennifer A. Rode, Eleanor F. Toye, and Alan F. Blackwell. The fuzzy felt ethnography – understanding the programming patterns of domestic appliances. *Personal Ubiquitous Computing*, 8(3-4):161–176, 2004.

[122] Michael Rohs. Real-World Interaction with Camera-Phones. In *2nd International Symposium on Ubiquitous Computing Systems (UCS)*, pages 39–48, Tokyo, Japan, November 2004.

[123] Michael Rohs and Jürgen Bohn. Entry Points into a Smart Campus Environment – Overview of the ETHOC System. In *IWSAWC '03: Proceedings of the 23rd International Conference on*

Distributed Computing Systems, pages 260–266. IEEE Computer Society, 2003.

[124] Michael Rohs and Christof Roduner. Camera Phones with Pen Input as Annotation Devices. In *Pervasive 2005 Workshop on Pervasive Mobile Interaction Devices (PERMID)*, pages 23–26, Munich, Germany, May 2005.

[125] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[126] Kay Römer, Thomas Schoch, Friedemann Mattern, and Thomas Dübendorfer. Smart Identification Frameworks for Ubiquitous Computing Applications. *Wireless Networks*, 10(6):689–700, December 2004.

[127] George Roussos, Juha Tuominen, Leda Koukara, Olli Seppala, Panos Kourouthanasis, George Giaglis, and Jeroen Frissaer. A Case Study in Pervasive Retail. In *Proceedings of the 2nd International Workshop on Mobile Commerce (WMC 2002)*, pages 90–94, Atlanta, GA, USA, September 2002. ACM Press.

[128] Aviel D. Rubin and Daniel E. Geer. Mobile Code Security. *IEEE Internet Computing*, 2(6):30–34, 1998.

[129] Enrico Rukzio, Gregor Broll, and Sergej Wetzstein. The Physical Mobile Interaction Framework (PMIF). Technical Report LMU-MI-2008-2, Lugwig-Maximilians-Universität München, Munich, December 2008.

[130] Enrico Rukzio, Karin Leichtenstern, Vic Callaghan, Paul Holleis, Albrecht Schmidt, and Jeannette Chin. An Experimental Comparison of Physical Mobile Interaction Techniques: Touching, Pointing and Scanning. In *UbiComp '06: Proceedings of the 8th International Conference on Ubiquitous Computing*, volume 4206 of *Lecture Notes in Computer Science*, Orange County, CA, USA, September 2006. Springer.

[131] Iván Sánchez, Marta Cortés, and Jukka Riekki. Controlling Multimedia Players using NFC Enabled Mobile Phones. In *Proceedings of the 6th International Conference on Mobile and Ubiq-*

uitous Multimedia (MUM 2007), pages 118–124, Oulu, Finland, December 2007. ACM.

[132] Jari T. Savolainen, Harri Hirvola, and Sassan Iraji. EPC UHF RFID Reader: Mobile Phone Integration and Services. In *Proceedings of the 6th IEEE Consumer Communications and Networking Conference (CCNC 2009)*, pages 1–5, Las Vegas, NV, USA, January 2009.

[133] Jürgen Scheible and Ville Tuulos. *Mobile Python: Rapid Prototyping of Applications on the Mobile Platform.* John Wiley & Sons, Chichester, England, December 2007.

[134] Christian Schmitt, Kai Fischbach, and Detlef Schoder. Enabling Open Innovation in a World of Ubiquitous Computing. In *Proceedings of the 1st International Workshop on Advanced Data Processing in Ubiquitous Computing (ADPUC 2006)*, Melbourne, Australia, November 2006. ACM.

[135] Beat Schwarzentrub. Interacting with Appliances Using Mobile Phones. Master's thesis, Institute for Pervasive Computing, Department of Computer Science, ETH Zurich, Zurich, Switzerland, 2007.

[136] Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open Mind Common Sense: Knowledge Acquisition from the General Public. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE, Proceedings*, volume 2519 of *Lecture Notes in Computer Science*, pages 1223–1237. Springer, 2002.

[137] M. A. Smith, D. Davenport, H. Hwa, and T. Turner. Object AURAs: A Mobile Retail and Product Annotation System. In *EC '04: Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 240–241, New York, NY, USA, May 2004. ACM Press.

[138] Sun Microsystems. Jini Architectural Overview. `www.sun.com/software/jini/whitepapers/architecture.pdf`.

[139] Frédéric Thiesse, Christian Floerkemeier, Mark Harrison, Florian Michahelles, and Christof Roduner. Technology, Standards, and

Real-World Deployments of the EPC Network. *IEEE Internet Computing*, 13(2):36–43, March 2009.

[140] Art Thomas and Ron Garland. Grocery shopping: list and non-list usage. *Marketing Intelligence & Planning*, 22(6):623–635, 2004.

[141] UDDI. UDDI Technical White Paper, 2000. `www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`.

[142] UPnP Forum. `www.upnp.org`.

[143] Felix von Reischach, Dominique Guinard, Florian Michahelles, and Elgar Fleisch. A Mobile Product Recommendation System Interacting with Tagged Products. In *Proceedings of the 7th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2009)*, pages 1–6, Galveston, TX, USA, March 2009.

[144] Jim Waldo. The Jini Architecture for Network-Centric Computing. *Communications of the ACM*, 42(7):76–82, July 1999.

[145] Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging Physical and Virtual Worlds with Electronic Tags. In *CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 370–377, Pittsburgh, PA, USA, May 1999. ACM Press.

[146] Alexander Yates, Oren Etzioni, and Daniel Weld. A Reliable Natural Language Interface to Household Appliances. In *IUI '03: Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 189–196, Miami, FL, USA, January 2003. ACM Press.

[147] Gottfried Zimmermann, Gregg Vanderheiden, and Al Gilman. Prototype Implementations for a Universal Remote Console Specification. In *CHI '02: CHI '02 Extended Abstracts on Human Factors in Computing Systems*, pages 510–511, Minneapolis, MN, USA, April 2002. ACM Press.