LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
Department “Institut für Informatik”
Media Informatics

Diploma Thesis

Mobile Interaction with Real World Services

Gregor Broll
gregor.broll@ifi.lmu.de

Duration:	01.01.2006 – 30.06.2006
Supervisors:	Dipl.-Inf. Enrico Rukzio, Massimo Paolucci (NTT DoCoMo Euro-Labs), John Hamard (NTT DoCoMo Euro-Labs)
Responsible Lecturer:	Dr. Albrecht Schmidt
Responsible Professor:	Professor Dr. Heinrich Hußmann

Abstract

This diploma thesis presents a generic approach to complex *Physical Mobile Interaction* that is based on the exploitation of *Semantic Web Services*. In this context, a framework is introduced that uses the flexibility and expressiveness of Semantic Web Service descriptions for the automatic and dynamic generation of adaptable user interfaces that support and facilitate Physical Mobile Interaction. Thereby, this diploma thesis focuses on the design, implementation and evaluation of a prototype client application that builds upon this framework in order to improve the mobile interaction with physical objects and provide more intuitive means for interactions with associated digital information and services. Based on the results of a preliminary user study and rapid prototyping which confirmed the overall approach, a prototype application was designed and implemented in order to be used with use case scenarios for mobile ticketing. For that purpose, the client application intermediates between Semantic Web Services and Physical Mobile Interaction on a generic level, as it exploits exchangeable Physical Mobile Interaction techniques for the invocation of different services within the framework for their mutual benefit. While the Web Services and their descriptions provide a rich background for complex Physical Mobile Interactions on one hand, their invocation is facilitated through the more familiar mobile interaction with physical objects on the other hand. The current implementation of the mobile client application supports the selection of physical objects through *Touching*, based on Near Field Communication technology, *Pointing*, based on the recognition of visual markers, and *Direct Input* that is implicitly supported by common interface widgets.

Zusammenfassung

Die vorliegende Diplomarbeit stellt einen allgemeinen Ansatz für komplexe physikalische, mobile Interaktionen (*Physical Mobile Interaction*) vor, der auf der Nutzung von *Semantic Web Services* basiert. In diesem Zusammenhang wird ein System vorgestellt, das die Flexibilität und Ausdruckskraft von Beschreibungen von Semantic Web Services für die automatische und dynamische Erzeugung von anpassungsfähigen Benutzeroberflächen nutzt, die physikalische, mobile Interaktionen unterstützen und erleichtern. Diese Diplomarbeit konzentriert sich dabei auf das Design, die Implementierung und die Evaluierung eines mobilen Anwendungs-Prototypen, der auf diesem System aufbaut, um mobile Interaktionen mit Objekten zu verbessern und intuitivere Möglichkeiten für die Interaktion mit dazugehörigen Informationen und Diensten bereitzustellen. Aufbauend auf den Ergebnissen einer vorausgegangen Nutzerstudie inklusive eines einfachen Prototyps, die diesen Ansatz untermauert, wurde ein Anwendungs-Prototyp entworfen und implementiert, um im Zusammenhang mit Nutzungs-Szenarien für mobilen Ticketkauf verwendet zu werden. Dazu vermittelt der Anwendungs-Prototyp zwischen Semantic Web Services und physikalischen, mobilen Interaktionen auf einer generischen Ebene. Dabei nutzt er austauschbare Techniken für physikalische, mobile Interaktionen, um verschiedene Dienste innerhalb des Systems zum beiderseitigen Nutzen aufzurufen. Während Web Services und ihre Beschreibungen einerseits eine wertvolle Grundlage für komplexe physikalische mobile Interaktionen bieten, wird ihre Nutzung durch die vertraute, mobile Interaktion mit realen Objekten erleichtert. Die aktuelle Implementierung der mobilen Endnutzer-Anwendung ermöglicht die Auswahl von Objekten durch *Berühren (Touching)*, basierend auf Near Field Communication Technologie, Zeigen (Pointing), basierend auf der Erkennung von visuellen Markern und *Direkte Eingabe (Direct Input)*, die allgemein von Elementen von Benutzeroberflächen unterstützt wird.

Task Definition for the Media Informatics Diploma Thesis "Mobile Interaction with Real World Services"

Description: We currently see a big interest in interactions with mobile services which are provided by the real world. So far the corresponding user interfaces which consist of the interfaces provided by the mobile phone and the real world interfaces where developed from scratch for every new application or prototype. This is not feasible for the development of commercial services or products because of the complexity of this time consuming process. A solution for this is the automatic generation of dynamic user interfaces which are presented by the mobile device or by other displays and the provision of guidelines for the design of the static real world user interfaces. This thesis should analyse the idea of automatically generating user interfaces for mobile interactions with the real world taking the context and the offered services into account, both represented by semantic descriptions. The goals of this work are:

- the analysis of corresponding technologies and interaction techniques
- the definition of an approach for automatically generating user interfaces for physical mobile interactions
- the integration of this approach in a framework for supporting physical mobile interactions which is developed in parallel by an other diploma thesis
- the definition of guidelines for the design of real world user interfaces
- the usage and development of a framework for supporting physical mobile interactions jointly with an other master thesis
- the evaluation of the prototype (together with a related master thesis)
- participating on a publication describing the work done within this diploma thesis

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt habe, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, 03.07.2006 ----------

Gregor Broll

Table of Content

Chapter 1

Introduction and Motivation

Over the last few years, mobile devices and especially mobile phones have established themselves as means for accessing, collecting and processing digital information from the real world. They are the most widely adapted personal computing platform, support a wide range of platforms and technologies and are always connected to different speech and data networks including the internet. Despite some technical constraints, they are powerful clients that provide increasingly sophisticated methods for capturing a multitude of different information from their rich, dynamic and personalised context. The richer their means for mobile information access become, the more information can be captured from the physical world and the more interactions with its objects become possible.

While mobile devices and clients develop more an more sophisticated methods to access and process information, objects from the real world are increasingly linked with information that goes beyond their inherent amount of data. These physical objects become associated with different digital information and services by being augmented with different technologies like Bluetooth, visual or wireless markers. They become recognisable and even get their own, unique, digital identities.

The next step is the evolution of these developments into *Physical Mobile Interaction* [RWS 05] which allows mobile devices to access and use digital information and services by interacting with the physical objects they are associated with.

Physical Mobile Interaction uses and exploits the more natural and more familiar interaction with physical objects in order to provide intuitive access to associated digital information and facilitate the interaction with mobile services. It uses different technologies in order to realize different interaction techniques for selecting physical objects and associated information: Mobile devices can take pictures of visual markers [RoGf 04] in magazines or on posters and use this information for the automatic invocation of associated services [RSH 04]. In combination with wireless technologies like RFID [Want 06] or Near Field Communication (NFC) [nfc] Physical Mobile Interaction is increasingly gaining importance. It reduces mobile payment, identification or access control to simply swiping a mobile phone over a reader. NTT DoCoMo's i-mode FeliCa service for example combines mobile phones with built-in NFC-chips and a service framework based on i-mode [feli].

The development of Physical Mobile Interaction and the increasing interest in it from both academia and industry is met and advanced by the dissemination of another, related technology: The *Internet of Things* [Mel 03], in which everyday objects are uniquely identified through

wireless markers and have individual network references. This standardised way to identify and describe objects facilitates access to and interaction with them, for example through Physical Mobile Interaction.
Currently there are several approaches to Physical Mobile Interaction through different applications – which are nevertheless mostly very simple and proprietary. They are designed for a limited scope and simple interactions and don't provide generic concepts or tools for the common description of real world services and the Physical Mobile Interaction with them. An example for such a system is the Nokia Local Interactions Server which is a real-time web service that acts as a back end for RFID-based mobile interactions [nlis].

In this context, this diploma thesis *"Mobile Interaction with Real World Services"* addresses the development and improvement of mobile interaction with physical objects. One of its goals is to support more complex Physical Mobile Interactions, e.g. with objects from the Internet of Things and transfer the familiarity of interacting with them to the interaction with associated information and services in order to make it easier and more intuitive. It tries to shift the focus of interaction to physical objects by pushing features off mobile phones, mapping them to real world objects and thus turning them into rich ubiquitous interfaces for new and more complex interaction techniques.
This diploma thesis is part of the Perci-project (PERvasive ServiCe Interaction) [perc], a collaboration between the University of Munich, Department of Media Informatics [dmi] and NTT DoCoMo Euro-Labs [ntt] that is funded by the later. One objective of this project is the combination of Physical Mobile Interaction and *Web Services* [LeHe 06] for their mutual benefit. Especially Semantic Web Service technology promises to be a great resource for realizing more sophisticated Physical Mobile Interaction as it makes information and services available while enhancing their interoperability, extensibility, expressiveness and independence.
Taking this objective into account, this diploma thesis focuses on a generic approach that exploits the flexibility and expressiveness of Semantic Web Service descriptions for the automatic and dynamic generation of adapted interfaces that support and facilitate Physical Mobile Interaction. This process promises to be a generic, flexible and efficient means for supporting different Physical Mobile Interaction techniques. For that purpose, generic yet expressive service descriptions can be reused for generating multiple, different interfaces that are adapted to different mobile devices, target platforms, user profiles and interaction designs. On the other hand, Web Services can benefit from the combination with Physical Mobile Interaction, as it provides a more natural and intuitive way of interacting with them.
Figure 1.1 shows an early outline of the Perci-framework that has been developed in a collaborative effort together with Sven Siorpaes. It addresses key issues of the presented approach, like the support for Semantic Web Services, the generation of abstract and concrete user interfaces for different mobile platforms and the exploitation of context information. Most of the framework including Semantic Web Services, the generation of an abstract user interface and the development of mechanisms to communicate them to mobile devices has been developed by Sven Siorpaes (see his diploma thesis [Sior 06] for details).

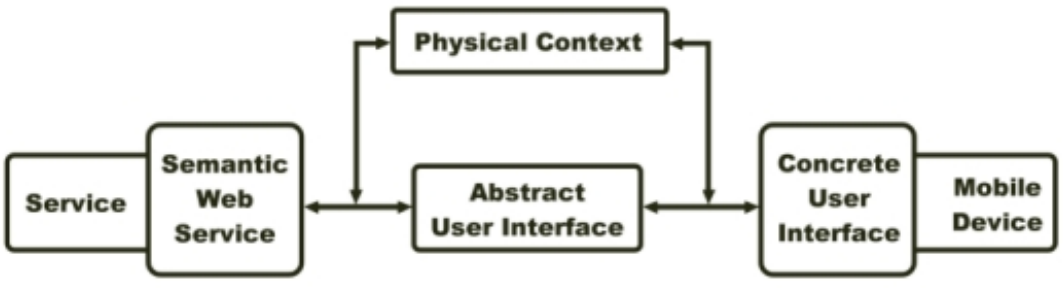


Figure 1.1: General overview of the Perci-framework (from [SRP+ 05])

Although this diploma thesis “Mobile Interaction with Real World Services” also deals with the design and development of the Perci-framework, it focuses on the realisation of a prototype mobile client application for Physical Mobile Interaction that builds upon this framework and uses it for the generation of a concrete user interface from Semantic Web Service descriptions.
In order to motivate and confirm the presented approach to combining Semantic Web Services and Physical Mobile Interaction, chapter 2 introduces and summarises different work from related domains. Chapters 3 and 4 describe a use case scenario for Physical Mobile Interaction and an early user study that was conducted with it. The Perci-framework and its architecture are the subject of chapter 5 which provides the background for the implementation of the prototype application in chapter 6. Chapter 7 concludes this thesis and gives an outlook to future work.

Chapter 2

Background and Related Work

This chapter introduces and summarises different related work that forms a background for the approach to Physical Mobile Interaction represented by the Perci-framework in order to motivate and confirm it. A special focus is put on interface description languages and the related automatic generation of interfaces since they provide an important foundation for central issues of the presented approach. Other chapters will familiarise Semantic Web Services, Physical Mobile Interaction or Context-Aware Computing in general.

2.1 An Introduction to Web Services

Web Services [LeHe 06] are ubiquitous, self-contained software systems and components that provide a standard means for supporting interoperability between different software applications. With Web Services, different applications running on heterogeneous platforms can interoperate over a network. Other advantages of Web Services apart from their interoperability are extensibility, location transparency and the exploitation of XML for machine-processable interface descriptions [Haas 06].

Web Services can be used as part of a *Service Oriented Architecture (SOA)*, a model that promotes the composition of independent, loosely coupled services. It establishes the basis for the interaction between Web Services and transfers them from a simple client-server model into a distributed system with three basic roles [Col 04]: *Service Requestors* invoke services offered by one or more *Service Providers* by exchanging and processing request and response messages. Service Providers can offer their own functionalities or act as Service Requestors themselves by aggregating functionalities of other Web Services into higher-level services using choreography languages such as the *Business Process Execution Language for Web Services (BPEL4WS)* [bpel]. Service Providers advertise their functionalities to the *Service Broker*, that acts as an intermediate between them and Service Requestors. It manages the publishing, location or discovery of services, e.g. using *Universal Description, Discovery, and Integration (UDDI)* [uddi], which implements a service registry and defines a standard interface for publishing and finding services based on *Simple Object Access Protocol (SOAP)* [MaLa 06] messages. Service Requestors ask the Service Broker for services whose functionalities meet their requirements and select the best service for interaction.

The interoperability between different Web Services is greatly supported by two standard technologies: SOAP is a simple, stateless XML-based protocol to let applications access Web Services and exchange structured and typed information over HTTP. On the other hand, the *Web Service Description Language (WSDL)* [CCMW 01] defines an XML grammar that describes Web Services as sets of communication endpoints as well as the input and output messages that are required for their invocation. WSDL specifies the types of exchanged messages, the binding to message formats and protocols and provides an interface for the invocation of Web Services by describing their basic operations. These operations are the building blocks of Web Services and provide the structure for the syntax and patterns of input and output messages [MPM+ 05]. Service Brokers publish the WSDL-descriptions of Web Service directly or register them using UDDI which facilitates service discovery through service properties.
Though providing a standardised way to define, discover, access and invoke Web Services in an uniform way, WSDL descriptions are restricted in several ways: They can only describe single processes and are dependent on additional languages such as BPEL4WS in order to define a workflow of several services. More important, WSDL describes services, their input and output only in a syntactical rather than a semantic way.
The *Web Ontology Language for Services (OWL-S)* [MBH+ 06] was developed in order to address these shortcomings and to provide a richer foundation for wider and more complex specifications of Web Services. OWL-S is an ontology for describing Semantic Web Services and is based on the *OWL Web Ontology Language* [McHa 04]. OWL itself is a XML-based language that uses the *Resource Description Framework (RDF)* [HSB 06] and *RDF Schema (RDF-S)* [HSB 06] for defining ontologies - formal specifications of meanings of objects respectively resources and the relationships between them. OWL is also part of the *Semantic Web* vision (see [Herm 06], [BLHL 01]) as it describes resources in a semantic, machine interpretable way.
The flexibility and expressiveness of OWL is also reflected in the way OWL-S describes services and how to invoke them. Opposite to WSDL, OWL-S not only supports single atomic processes, but also more complex composite processes that are composed of several atomic processes. This allows and facilitates automatic service invocation for the development of more dynamic and flexible service clients, automatic service discovery based on service capabilities and automatic composition of several (simple) services in order to cope with complex tasks [MBH+ 06]. Figure 2.1 shows the structure of an OWL-S ontology with the 3 main building blocks of its service description – the *Service Profile*, the *Service Grounding* and the *Service Model* - that are attached to the main *Service* class with the properties *presents*, *supports* and *described by* [MBH+ 06]:

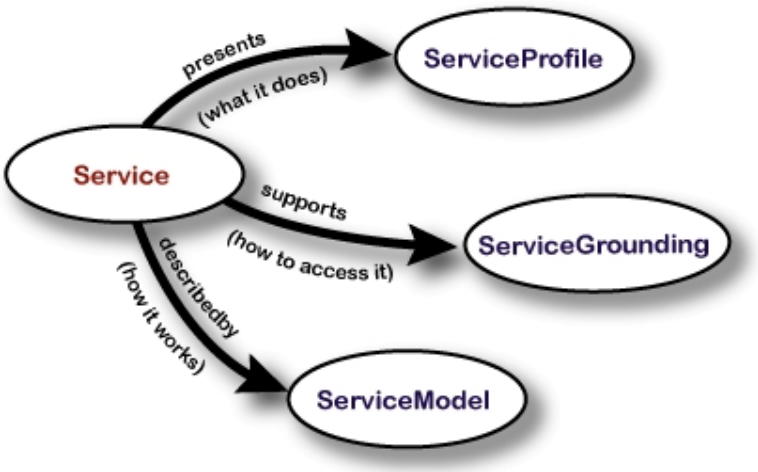


Figure 2.1: OWL-S ontology structure (from [MBH+ 06])

- The Service Profile describes what a service does and what properties and capabilities it offers to requesting clients. Among these properties are the type of the service, its functional specification such as inputs, outputs and preconditions, security requirements, quality of service ratings and requirements that prospective Service Requesters have to meet in order to use the service successfully.
- The Service Model or Process Model describes how a service and the internal processes that define its operations work. It facilitates the invocation of Web Services, their composition and the monitoring of their interaction and differentiates between atomic processes and composite processes. It describes how to use a service and what results can be expected from it by specifying the semantic content of inputs and preconditions of service requests as well as outputs, results and effects of a process.
- The Service Grounding builds upon WSDL and comprises detail about how clients can access Web Services and thus specifies communication protocols, transport mechanisms, message formats and addressing. Following its name, the Service Grounding maps each semantic type of input or output defined in the Service Model to concrete message specifications grounded in WSDL.

Web Services provide the foundation for the Perci-framework. Where Web Service technologies like WSDL are still constrained, Semantic Web Services provide enough expressiveness and flexibility to be exploited for the automatic generation of adapted user interfaces in order to support Physical Mobile Interaction.

2.2 User Interface Description Languages

User interface description languages (UIDL) are generic languages that describe user interfaces and are the basis for their dynamic and automatic generation. Their use is motivated by current trends and problems: Multiple, separate interfaces are developed for an increasing number of applications on different platforms with specific constraints (from PCs to web browsers to cell phones) using an abundance of target technologies. Developers often have to create separate interfaces for the same application on different platforms, using different languages from Java (Micro Edition) to XHTML, Flash or WML. In this context, UIDLs provide several advantages (see [AbPh 99]):

- distinct separation between interface and application code
- reduced development time
- better support for rapid prototyping
- easier internationalisation and localisation
- easy to learn and use, even without specific knowledge of underlying platforms or programming languages
- reusable for the development of the same interface for different devices and platforms

In the context of the presented approach to interface generation, UIDLs can help describe interfaces in a generic and platform independent way in order to make the process of interface generation more dynamic, flexible and adaptable to different context information. The Perci-framework could benefit from the advantages of UIDLs and use them as a step in the transition from Semantic Web Services and their descriptions to concrete user interfaces as they provide the foundation for the interface generation process.
Some UIDLs define interfaces on more abstract and generic levels in order to be reusable and independent from target platforms and technologies. Others describe interfaces in very specific terms and are often closely related to concrete technologies. This chapter introduces some example UIDLs and examines their purposes, differences and similarities in more detail.

2.2.1 The User Interface Markup Language (UIML)

UIML [uiml] is a XML-compliant markup-language for describing user interfaces in an abstract, declarative manner independent of different platforms, programming languages, operating systems or interface-paradigms. It has been developed and advanced since December 1997, mostly by Virginia Tech's Center for Human Computer Interaction [chci] and Harmonia Inc. [harm].

UIML was designed to describe and create user interfaces for applications on different devices and platforms outside the application code. Its derivation from XML supports the portability of interface descriptions between different platforms and guarantees extensibility for new interface-metaphors and target languages.

The UIML specification provides a simple markup whose tags are not specific to a particular toolkit or programming language but are generic in order to describe interface elements in an abstract way. Basically UIML-tags capture 3 things: They record the appearance of an interface and its elements, their positions, order and layout. UIML also defines the user interaction with an interface which is the type of behaviour that is usually recorded by event handlers (What happens when a user operates an interface-widget?). It also connects an interface to the application logic and its functionalities behind an interaction (What action is performed as result of an interaction?).

Figure 2.2 shows a code fragment of an UIML description which records interface elements and their appearance on a very generic level, so that they can be rendered for different target languages. The example describes a named frame-element along with position and size properties. The definitions of structure and style of a description are separated from each other so that different parts of the description can be rendered using different output technologies. Other UIML-elements describe the behaviour that is assigned to interface elements, different types of content that can be displayed with them (e.g. for internationalisation) or the mapping of elements to concrete interface widgets in a target language.

```
<structure>
        <part name="main" class="JFrame">
            <style>
                <property name="title">MyTitle</property>
            </style>
        </part>
</structure>

<style>
        <property part-name="main" name="bounds">0,0,500,300</property>
</style>
```

Figure 2.2: UIML code fragment (from [Stöt 01])

The transition from a generic UIML-description to an interface that consists of the elements of a specific target language is achieved by different renderers. Renderes are tools that allow efficient compilation of abstract UIML-descriptions and their translation into a chosen target language. Currently renderers are available to build interfaces for Java, HTML, WML and VoiceXML.

Although UIML itself is language- and toolkit-independent, it is not possible to create multiple interfaces for different target languages from the same UIML-description. Due to the differences in the vocabularies of target languages, developers still have to design separate interfaces for different target languages and tailor descriptions to the syntax supported by different rendering engines.

UIML interfaces can be deployed in several ways: It can be installed with an application as a rendered interface or as a raw UIML-description that has to be translated by an appropriate

renderer that is distributed with the application. UIML-descriptions can also be stored on a server from where applications can download it upon request (see Figure 2.3).

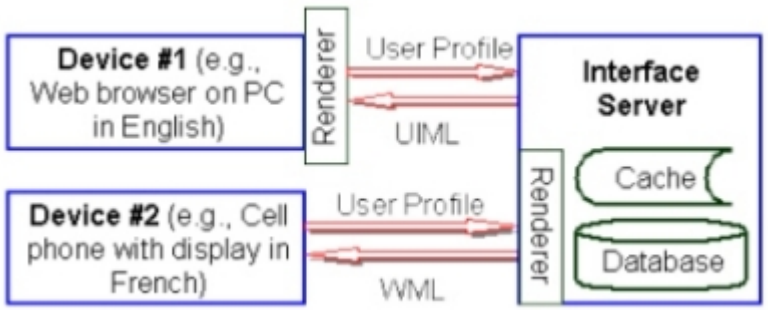


Figure 2.3: Deployment of UIML interface descriptions (from [AbPh 99])

Clients can communicate their specific requirements concerning their platforms and intended target language to the server via a user profile-description. Clients which provide their own renderer simply download a UIML-description and render the code themselves. That way clients only have to download a lean UIML-file instead of bulky executable code which also enhances network security. Clients that can't render UIML themselves due to their limited resources (e. g. insufficient processing power or limited memory on mobile phones) can ask the server to render the interface in the appropriate target language and send its code back.

2.2.2 XUL

XUL [xul] is a XML-based interface language for describing concrete graphical user interfaces. It started as a part of the Mozilla-project [moz] in 1998 and was originally developed to describe the graphical interfaces of Mozilla related applications like Firefox or Thunderbird whose entire GUI is created from XUL descriptions. The acronym "XUL" is often translated as "XML User Interface Language" but its pronunciation as "Zuul" is a reference to the movie "Ghostbusters" where Zuul is the name of the ghost of an ancient Sumerian god (see [wiki a]).
XUL is a XML-compliant markup language that supports and integrates other XML related technologies and separates between content and style. CSS style sheets define the layout and appearance of interfaces which facilitates the creation of different skins. JavaScript is used to define behaviour in XUL interfaces. The XUL interface toolkit also supports the use of XHTML, SVG, RDF, XSLT, XPath, Soap or WSDL (see [xulp]).
XUL interface descriptions define elements and layout of sophisticated user interfaces with rich sets of different components and widgets on a very concrete level. A XUL interface is usually described as three discrete sets of components: The XUL document itself which defines the layout of the user interface (content), the CSS and image files that define the appearance of an XUL application (skin) and several DTD documents which define entities for easy software localisation (locale) (see [wiki a]).
Since XUL supports DOM technology, the central object of a XUL file is the Document Object which describes the XUL document itself. XUL interface descriptions themselves comprise a wide range of elements from typical web applications as well as several elements and widgets from traditional desktop applications (e.g. window, dialog, text box, list box, radio button, toolbar). Figure 2.4 shows a short XUL code example and the interface rendered from it.
XUL is not restricted to Mozilla application interfaces but can also be used for other software. Together with its rendering engine Gecko, XUL forms a powerful framework for the development of applications that require a rich, dynamic user interface. An example for a browser-related application is the Mozilla Amazon Browser [mab], a web client application that uses a complex XUL interface to search products from Amazon.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window
    id="sample-window"
    title="Beispiel"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
      <vbox>
          <checkbox label="CheckBox"/>
          <hbox>
              <spacer flex="1"/>
              <button label="OK"/>
              <button label="Abbrechen"/>
          </hbox>
      </vbox>
</window>
```

Figure 2.4: Example of XUL code and the rendered interface (from [wiki a])

The evolution of XUL moves from browser-based applications to the creation of interfaces and standalone applications that are independent and portable across system boundaries. Although the derivation from XML makes XUL independent to some degree, it is still necessary to bring the entire Mozilla Toolkit [moto] to a target platform in order to run XUL applications. On the other hand, this toolkit contains all necessary technologies to run and develop XUL applications including the Gecko rendering engine as well as support for CSS or JavaScript.
The efforts to make XUL more platform independent have been channelled into projects such as XULRUNNER [xulr] which aims at creating a GRE (Gecko Runtime Environment) [gre] based on a separate, Mozilla independent Gecko rendering engine. This would greatly support the development of standalone applications based on a framework of Gecko and XUL and especially increase the cross-platform-portability of such applications.

2.2.3 The User Interface Extensible Markup Language (USIXML)

USIXML is a XML-compliant language that has been developed as part of the Cameleon Project [capr]. It comprises a declarative language to describe characteristics of an interface, such as design, interaction techniques or modality of use. It focuses on the specification and generation of interfaces on different levels of detail and abstraction while maintaining the mappings between these levels. Another focus of USIXML is the description of interfaces for multiple contexts of use such as character-, graphical-, auditory- or even multimodal-user interfaces [VLMB+ 04].
USIXML is structured according to the 4 levels of abstraction defined by the Cameleon Reference Framework (Figure 2.5). The framework defines different steps in the development life-cycle of interfaces for multi-context-sensitive interactive applications [VLMB+ 04].

- The *Task & Concepts* level describes one or several (interactive) tasks that are carried out by a user as well as various objects that are manipulated by these tasks.
- The level of the *Abstract User Interface (AUI)* is independent of any interaction modality (e.g. graphical, vocal, speech synthesis) and describes an abstract user interface definition for the Concrete User Interface-level.
- The *Concrete User Interface (CUI)* allows the description of appearance and behaviour of an interface and its elements that can be perceived by users. An interface on this level is dependent on a certain modality but is independent of any computing platform
- The *Final User Interface (FUI)* is related to an operational interface that is running on a special computing platform either by interpretation (e.g. through a web browser) or by execution after the compilation of code in an interactive development environment.

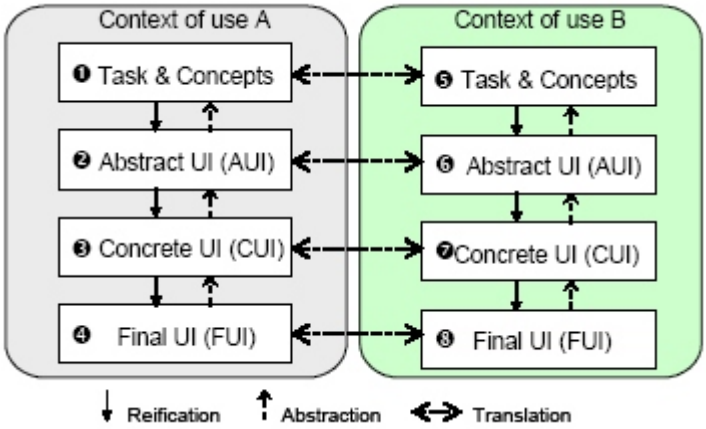


Figure 2.5: The Cameleon Reference Framework (from [VLMB+ 04])

The Cameleon Reference Framework supports 3 types of transformations between the abstraction levels which supports multidirectional user interface development. *Abstraction* and *Reification* are opposite processes but both of them elicit artefacts that are more abstract (respectively more concrete) than the artefacts that serve as input to the process. *Translation* on the other hand elicits artefacts intended for a particular context of use from artefacts of a peer development step that is aimed at a different context of use.
The collection of USIXML models that describe an interface is structured according to the 4 basic level of abstraction defined in the Cameleon Reference Framework in order to represent the different steps in the development life cycle of context-sensitive interactive applications. Figure 2.6 shows an example of these steps and how each level of the framework can be decomposed into two sublevels of a hierarchical order which increases the possible graduation of abstraction:

- The *FUI level* is divided into a code sub-level and a rendering sub-level. The rendering defines how a certain interface or a part of it (e.g. a download-button) which is coded in a certain language is rendered depending on the underlying toolkit, platform or presentation manager. In the example a button is uniquely specified by HTML on the code level but can look different on the rendering level if rendered on MacOS X or Java Swing.

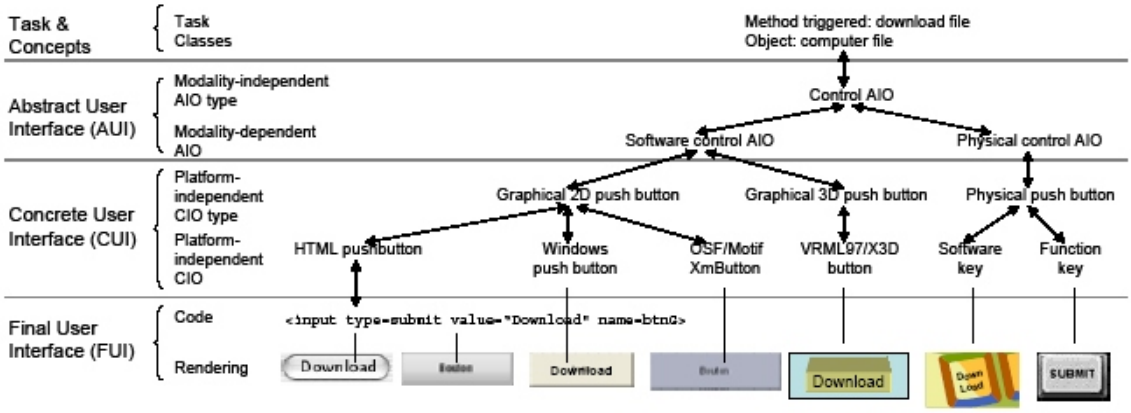


Figure 2.6: USIXML transformations (from [VLMB+ 04])

- The *CUI level* abstracts the FUI level independently of the computing platform and is therefore decomposed into the sub-levels platform-independent CIO and the more generic CIO type. Continuing the example from above the CIO element “HTML push button” belongs to the more generic CIO type “Graphical 2D push button” along with the “Windows push button” and the “OSF/Motif push button”.

- Just as the CUI level abstracts the FUI level regarding platform-independence, it is itself abstracted by the *AUI level* regarding the independence from any modality of interaction. The AUI level is decomposed into modality-independent AIO and AIO type. Examples are “Software control AIO” and “Physical control AIO” which both belong to “control AIO”.
- The *Task & Concepts* level is subdivided into tasks that should be performed using an interface (like downloading a file) and the classes whose objects are manipulated during those tasks (like a computer file that is downloaded).

With this hierarchy of abstractions it is possible to define mappings between instances and objects at the different levels and to develop transformations that find abstractions, reifications or combinations of both. Combinations of transformations allow establishing development paths for multi-directional user interface development.

2.2.4 Review of UIDLs

Despite the mentioned advantages of UIDLs, neither UIML, XUL, USIXML nor related interface description languages seem to be suitable for the presented, specific approach for several reasons:

- The intention of UIML is not yet reached. Although it provides a toolkit- and platform-independent approach to describe interfaces, it is too inflexible and not generic enough. It can’t use one single interface description for the generation of multiple interfaces due to the differences in UIML renderer syntaxes. Still different UIML descriptions have to be used to adopt interfaces for different renderers in order to generate multiple interfaces for different languages.
- XUL provides a set of interface descriptions that are too concrete, too close to browser- and desktop-applications and too inflexible in order to be useful for applications in the mobile domain. Additionally it needs the Mozilla Toolkit in order to work on any platform.
- While XUL is too concrete, USIXML is too general and puts too great a focus on multi-dimensional user interface development which is supported by its hierarchy of abstraction levels, the transformations between them and the description of different domains of interface-related information through different models. For these reasons UXISML rather seems to define a framework for interface development than a concrete language for building real world interfaces.

The evaluated UIDLs are mostly either too abstract or too heavyweight, don’t provide sufficient support for mobile user interface or have no connection to Web Services. Since they don’t fit the scope and the requirements of the Perci-framework’s approach to interface generation, they can only serve as drafts for the creation of a more suitable abstract user interface description.

2.3 Automatic Generation of User Interfaces

UIDLs and the automatic generation of user interfaces naturally go hand in hand. UIDLs need to be transformed and rendered from abstract interface descriptions into specific user interface widgets for different target platforms. The interface generation process on the other hand needs templates from which to create interfaces, preferably while considering context information about target platforms, devices and their users. The Perci-framework applies automatic interface generation in order to mediate between Web Services and mobile devices for Physical Mobile interaction in order to balance the efficient use of service descriptions and their exploitation for supporting different target technologies. Therefore this chapter introduces several approaches to the automatic generation of user interfaces.

2.3.1 Pebbles and PUC

The Pebbles project (PDAs for Entry of Both Bytes and Locations from External Sources) [Mye 05] of the Carnegie Mellon University explores the communication between mobile devices, personal computers, computerized appliances (such as radios, VCRs or stereo sets) or factory equipment.

As a part of Pebbles, the Personal Universal Controller (PUC) project [Nich 04] investigates how mobile devices like PDAs or mobile phones can be used as “remote controls” for common electrical appliances and how to improve the interaction with them. In its approach, appliances are equipped with abstract specifications of their functionalities. Mobile devices can download these specifications on demand and use them to automatically generate user interfaces for controlling the corresponding appliances. The interaction between client device and appliance is handled using two-way-communication which allows users to both send control messages to the appliance and receive feedback on their actions. The automatic interface generation in PUC enables different interface modalities (graphical and speech) on different platforms (PDA, mobile phone, smartphone, desktop computer). It also supports customisation to user preferences, consistency with other, similar interfaces and the combination of controls for multiple connected appliances into a single interface.

Figure 2.7 gives an overview for the PUC architecture and its main components. Apart from the supported communication technology (e.g. IEEE 802.11 or Bluetooth) for downloading appliance specification or exchanging control messages and feedback, the architecture has 4 main parts: appliance adaptors, specification language, communication protocol and interface generators [Nich 04].

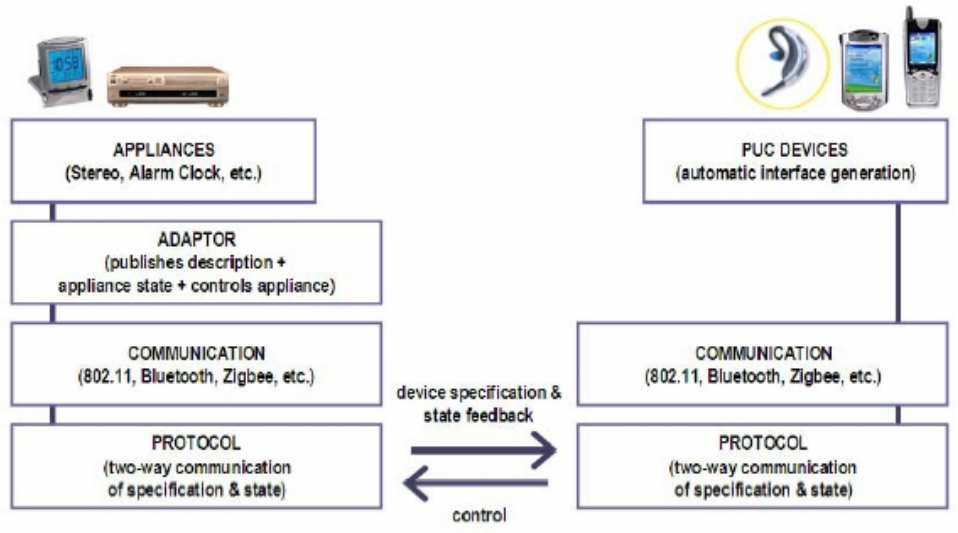


Figure 2.7: The PUC Architecture (from [Nich 04])

The automatic generation of user interfaces in PUC is based on the abstract description of appliance functionalities through a specification language. The language used in PUC was not designed to include specific information about the look and feel of an interface but only for its generation. For that purpose it includes state variables and commands for describing the functions of an appliance, a hierarchical "group tree" to organize the interface, information describing the availability of states and commands in regard to dependencies among each other as well as multiple strings for interface labels.
The communication protocol used in PUC allows controller devices to download interface specifications from appliances. Being a two-way-communication protocol it supports better interaction between the controlling device and the appliance as users receive feedback from the appliance about the success of their actions and the state of the appliance upon sending their control messages to them.
Since appliances don't natively support this communication protocol but their own, proprietary one, adaptors have to be build for each appliance to translate between the two protocols and enable interoperability with the system. In the context of PUC a number of adaptors for different appliances have been built, among them some general purpose adaptors for industry standards such as UPNP [upnp].
After the specification has been downloaded from an appliance onto the controlling device, an interface generator uses this specification to generate a remote control user interface for the platform of the device. Within PUC several interface generators have been build supporting different target platforms including graphical interfaces for PocketPC, Microsoft Smartphones and desktop computers.

2.3.2 SUPPLE

Supple is an application and device-independent system for the automatic generation of user interfaces for ubiquitous applications and is currently under development at the University of Washington (see [supp]). The idea behind SUPPLE is to enable applications to automatically generate and adapt their interfaces for a wide variety of devices, form factors, interaction modalities and user preferences.
Contrary to other approaches like the model-based interface-rendering of Pebbles (see previous chapter), SUPPLE uses decision-theoretic optimisation to generate interfaces from 3 different abstract functional specifications (see [GaWe 04]):

- A functional interface specification which depicts the types of data that are to be exchanged between the user and an application. It is usually provided by an application itself or from an appliance developer.
- A device model describes the capabilities of a display device and its widgets and is usually provided by the manufacturer of that device. It includes a cost function that is used to estimate the user's effort that is required to operate the available widgets using the interaction methods that are provided by a device.
- A user model which includes typical activities of a user modelled as a device- and rendering-independent user trace. This information can be used to automatically adapt user interfaces to different task and work styles. It is understood to be stored in some kind of personal information space, e. g. a personal server.

The information from these three models is combined by an algorithm that tries to assign interface widgets to the single elements in the interface specification. The greatest challenge for this algorithm is to find the best combination of suitable widgets among all possible combinations that meets all device constraints, pays attention to the traces contained in the user model and especially minimizes the device model's cost function that is derived from the types of widgets and modalities of interaction a device supports.
SUPPLE's goal to provide interfaces for ubiquitous computing applications is reflected by the distributed organisation of its architecture that regards the mobility of users and applications, the constraints of mobile devices and the spontaneity of interactions (see Figure 2.8). All three

models of the SUPPLE system could reside on different, physically separated devices and none of them could be powerful enough to run the algorithm for rendering the interface. These constraints and the distribution of components lead to the modularised architecture of SUPPLE that is shown in Figure 2.8. As none of the single components that hosts one of the three models could provide the computational power to run the interface rendering algorithm, another more powerful computer is employed for this task. That way the three models for the user, device and interface can be sent to the computer running the SUPPLE algorithm which takes these models as input for rendering the interface (see arrows 1,2,3 in Figure 2.8). The rendered interface is sent back to the display device for operation (arrow 4). The dashed lines in Figure 2.8 show the flow of data while the user interacts with the interface. During this interaction his actions are communicated to the application for execution and the resulting changes in the state of the application are returned to the interface for feedback (arrow 5). A user's interactions with an application may then be routed to his personal information space for updating the hosted user model including the user's trace of actions.

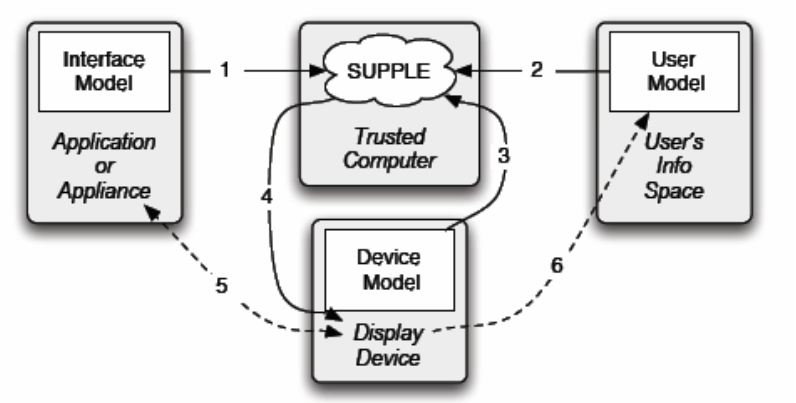


Figure 2.8: The SUPPLE architecture (from [GaWe 04])

Figure 2.9 shows how the SUPPLE system incorporates different device constraints and interaction modalities into the rendering of different interfaces for the same application - a system for switching lights in a classroom. Although both interfaces in Figure 2.9 have been rendered for devices with the same display size they differ in the types of widgets and their layout. This is because SUPPLE regards the different methods of interaction for each of these devices supports in order to render interfaces that cause its users the least effort to operate – be it pointer- or touch panel-based.

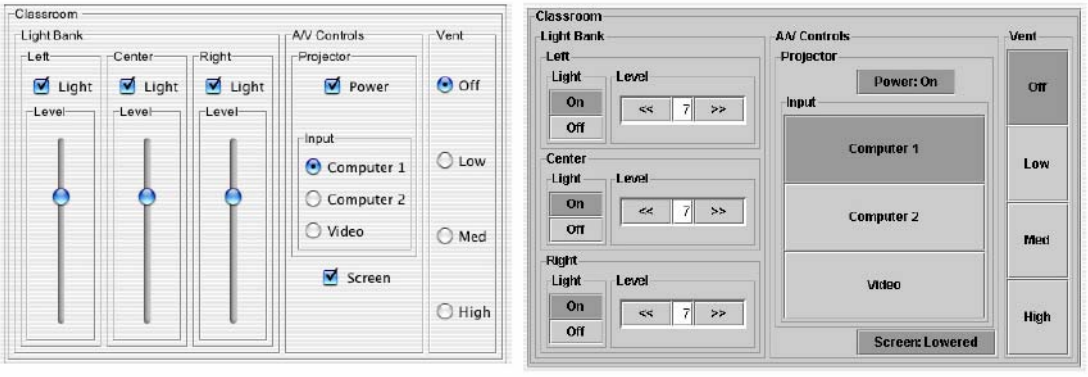


Figure 2.9: A classroom lighting-switching interface rendered for different devices under different constraints (from [GaWe 04])

2.3.3 An Ontological Approach to Generating User Interfaces for Web Services

As common interface description formalisms are not expressive enough to communicate the semantics of Web service features, Deepali Khushraj and Ora Lassila [KhLa 05] introduce an approach to enhance and extend Web Service descriptions using Semantic Web technology. Semantic Web Services annotated with OWL-S [MBH+ 06] ontologies provide semantic descriptions of how they work including inputs, outputs, process flow as well as pre- and post-conditions.

Khushraj and Lassila use semantic descriptions from the OWL-S Service Profile and Process Model as a basis for the generation of dynamic form-based user interfaces. As some of their properties can not be derived from these descriptions, Khushraj and Lassila extend Web Service ontologies with additional user interface annotations. Each OWL-S service description is associated with a UIModel extension which includes information about display labels, preferred widget types for providing the needed input or the grouping of fields and sub-fields.

Apart from automatically generating interfaces from Semantic Web Service descriptions, Khushraj and Lassila also suggest using additional information about the user for the personalisation of the interfaces. The type information of Web Service inputs and outputs can be associated with information the system has about its users and this relationship can be exploited to personalise interfaces and adapt them to the individual context and preferences of their users. For example the system could provide pre-selected options for widgets, tailoring the type and number of possible choices to those that have a significance for the user, his context and his actions.

The main part of the architecture for the automatic generation of dynamic, personalized user interface is a *Web Service Proxy* that mediates between the client and the Semantic Web Services (see Figure 2.10). When the client wants to invoke a Web Service, the Rendering Engine of the Web Service Proxy uses the OWL-S description and the UIModel extension of that service for rendering a web-based interface that can be based on HTML, XHTML or XForms.

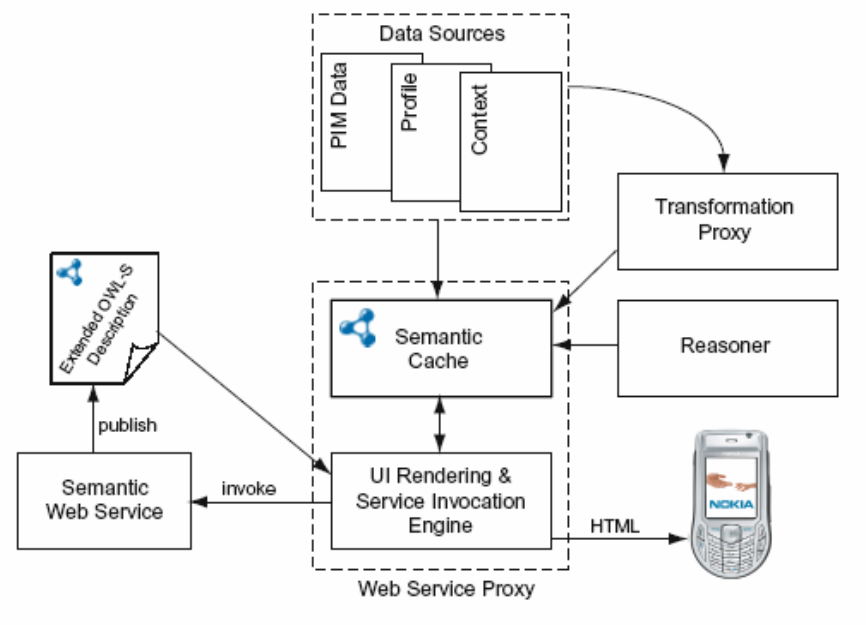


Figure 2.10: Architecture and system components (from [KhLa 05])

The system exploits both explicit and implicit relationships between user-related information from the *Semantic Cache* for rendering a personalised, context-aware interface. The Semantic Cache collects information about users from different data sources such as personal profiles,

context and input histories or PIM-data. Additionally a *Transformation Proxy* is used to support the collection of data from legacy systems and a *Reasoner* infers implicit relationships between the data collected in the Semantic Cache. All data in this repository is represented using a semantic model and has associated type information, which is used by the rendering algorithm to determine the relevance of that data. During the rendering of the interface this type information is mapped to the information types of the inputs and outputs of the services.
After the rendering, the personalised interface is displayed on the client device. The user interacts with the interface, makes inputs and upon calling the service, a Service Invocation Engine that is also part of the Web Service Proxy routes requests from the client to the according Web service. After its invocation the Web service returns an output which is again display on the client user interface.

2.3.4 Review of Automatic Interface Generation

While the presented approaches to the automatic generation of mobile interface from Pebbles, SUPPLE and [KhLa 05] provide an interesting background for the Perci-framework, [KhLa 05] is the one that is closest to its requirements and scope. Similarly, Khushraj and Lassila exploit Semantic Web Service descriptions based on OWL-S as a foundation for the generation of interfaces and utilizes user context information for this process. Although it does not provide the same sophisticated, distributed architecture as SUPPLE or generates as wide a range of different interface as Pebbles and SUPPLE, it confirms the approach of the Perci-framework and acts as its role model.

2.4 Context Definition and Description

Context-Aware Computing is an important part of Mobile Computing as mobile devices can make substantial use of the rich and dynamic information from their mobile and personalized context in order to provide new services (e.g. location based) and soften their technical constraints. In the context of the Perci-framework, context information about target platforms, mobile devices, users and their environment provides valuable input for the interface generation process and enhances its flexibility and adaptiveness. This chapter gives a rough overview of several different ways of how to define, use and describe context information.

2.4.1 Definition and Categorisation of Context

The term „*context*“ was first introduced by Bill Schilit and Marvin Theimer in 1994 [ScTh 94] as information about location, identity of people or objects or as changes to them. Since then many different definitions of context have been created reflecting many different areas of application and research. Nevertheless context is not clearly defined but more or less commonly understood as information about the environment of some entity – depending on the topic and focus of research or application.
Probably the most systematic and most common definition of context comes from Dey and Abowd [DeAb 99]. They analysed and compared previous definitions which described context through synonyms or enumerating examples for context information. Dey and Abowd developed their own definition of context as a summary and abstraction of these definitions and emphasized the ubiquitous character of context information and its relevance to the described entity [DeAb 99]: *„Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”*.
Instead of being defined, context is often only categorized which is helpful to identify different context types especially as parts of a concrete application. Context information can be categorized on different levels of abstraction to gain an overview and to order the different types of context systematically.

Apart from their rather abstract definition of context, Dey and Abowd categorize context information as *"primary context"* and *"secondary context"* [DeAb 99]. Primary context of an entity comprises the most frequently used types of context, namely location, identity, activity and time. Primary context can also be seen as an index to secondary context as e.g. the address, telephone number or birth date (secondary context) can be derived from the identity (primary context) of a person (the entity).
Other categorizations categorize context information into different classes depending on the focus of the described application and interest of research. Schmidt, Beigl and Gellersen [ScBG 98] divide context information into *„human factors"* and *„physical environment"* on the first level of their hierarchy. On the second level *„human factors"* is divided into the categories *„user"* (e.g. habits), *„social environment"* (e.g. relationships with other users) and *„task"* (e.g. activities) while *„physical environment"* is divided into *„location"* (e.g. relative and absolute positioning), *„infrastructure"* (e.g. hardware) und *„conditions"* (e.g. temperature).
Simple items of context information can be collected, combined or transformed into more valuable high-level context which forms its own domain within the definition and categorization of context. For example items of context information can be assembled over a certain time to form a context history, a chronologically ordered collection of information or actions from which patterns or habits can be derived. Simple GPS-coordinates can be refined to provide more abstract high-level location information, e.g. the name of a street. Another example of high-level context would be the combination of location, user preferences and database information to give a user recommendations for restaurants in his vicinity.

2.4.2 Definition and Categorisation of Context-Awareness

Context-Awareness basically describes the ability of an application to use context information in order to react to changes in its environment or to ease the interaction with it. Parallel to context, there are many definitions and categorizations of context-awareness and context-aware functionalities which applications can support.
Similar to their definition of context, Dey and Abowd compared several previous descriptions and definitions of context-awareness according to which applications would either use context information directly or adapt to it indirectly. Their own definition is again a summary and abstraction of its more specific predecessors (see [DeAb 99]): *"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."*
Categorizations of context-awareness mostly describe classes of functionalities and features of context-aware applications: Dey and Abowd propose 3 categories of context-sensitive functionalities that applications can support (see [DeAb 99]). Depending on the context, information or services that could be beneficial for the user are passively presented or highlighted (*"presentation"*). Information about the current context - e.g. all printers in an office and their availability - are merely presented and their usage is left to the user. Applications can also be used proactively according to the current context, e.g. automatically sending a document to the nearest available printer (*"automatic execution"*). Ultimately users can attach their own information to a context, augment it by *"tagging"* and associate both information for himself and others. For example a user can stick a virtual note to a sink of dirty dishes in his kitchen, so that other users are reminded to clean them.
Schmidt, Beigl and Gellersen on the other hand identify three domains that especially support mobile applications (see [ScBG 98]): *„Context-aware communication"* uses context information to control automated communication and filter information, e.g. in a context-aware todo-list. *„Context-aware application selection"* specifies the proactive choice of applications depending on the context in which it is mostly used. As users enter a telephone box, their context-aware mobile phone would automatically display their address book – an enhancement especially for mobile devices which are constrained regarding their usability. In the same context *„context-aware ui modes"* support the adaptation of user interfaces of mobile devices to the current context. Their usability could be increased by adapting the font-size of their displays

to constraints of the environment like the level of brightness or the speed of the user's motion (smaller font when walking, bigger font when running).

2.4.3 Describing Context Information

The meaningful description of context information is a prerequisite for its usage in a context-aware application as suggested in the previous chapters. Context information is provided by sensors that measure, collect or derive various data as well as their specific properties and changes. Sensor data comes from many different sources, among them sensors that measure physical parameters of the environment (e.g. temperature, noise or brightness), explicit input from users or even databases that provide details on device hardware specifications.
These sensor data are often raw and provided in a format proprietary to a sensor and have to be formatted and described in order to be useful for applications. The description of context information has to cover and respect its broad range of properties: Sensor data from which context information are derived are often imperfect, inconsistent and have different characteristics, e.g. temporal (static, dynamic). There are often alternative representations and different abstractions of context information possible as well as a high interrelation among them. Different approaches use different paradigms to model and describe context information, their properties and associations, e.g. as simple key-value-pairs, UML- or ER-diagrams, object oriented approaches or using XML.

CC/PP (Composite Capabilities/Preference Profile)

An example for a context description framework is CC/PP (Composite Capabilities/Preference Profile) a proposal of the W3C for a profile representation language [BHK 04] that is now managed by its Device Independence Activity Working Group [Frou 06]. The CC/PP framework is based on RDF [HSB 06] and is intended for the expression and description of device capabilities and user preferences. The specification of CC/PP defines a basic structure for profiles with different components having several attributes. Different vocabularies can be used to provide new components and attributes which ensures the extensibility of the framework. The Open Mobile Alliance [oma] developed a vocabulary, the WAG User Agent Profile (UAProf) that describes the profiles and capabilities of WAP [oma] devices.

A Context Description Framework for the Semantic Web

Another approach to modelling context information using RDF is presented by Oleksiy Khriyenko and Vagan Terziyan [KhTe 05]. They introduce the Context Description Framework (CDF) which extends RDF with capabilities to model highly dynamic and context-sensitive information.
The suggested CDF extends RDF statements logically and defines quadruple statements compared to RDF triples (see Figure 2.11). A CDF quadruple is a Statement that inherits from the *rdf:Statement* and has an additional *cdfs:trueInContext* property from the namespace of the CDF-Schema Vocabulary Description Language which is an extension of RDF-Schema [HSB 06]. The first three components of a CDF quadruple are similar to a RDF triple of subject, predicate and object. Subjects and objects of CDF statements are instances of *rdfs:Resource* and an CDF predicate is an instance of *cdfs:Property*.
The fourth component of a CDF quadruple is the context of the statement. A CDF statement may be true or false concerning other statements about its environment. Hence the context of a statement is defined as a set of other statements which describe the conditions of this environment. In Figure 2.11 these statements are collected in a (Contextual) Container and referenced by the cdfs:trueInContext-attribute of the according statement. Contextual statements can also be nested as they have their own contextual environment and conditions as well.

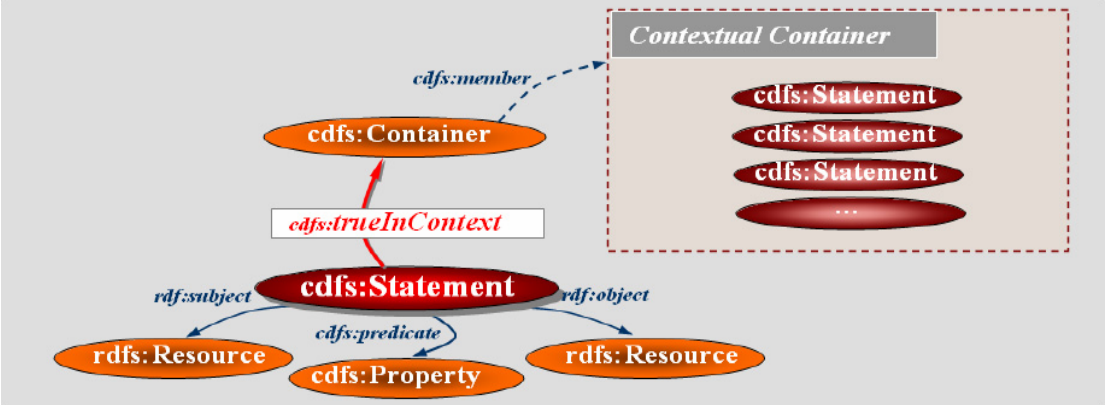


Figure 2.11: Graphic representation of a CDF quadruple (from [KhTe 05])

2.5 Physical Mobile Interaction

2.5.1 Definition and Examples

Physical Mobile Interaction [RWS 05] is a mobile interaction paradigm that evolved from the dissemination of mobile devices that provide more and more powerful means to access, capture and process information from the real world, as well as its objects, that are increasingly augmented and associated with digital information. Hence Physical Mobile Interaction employs mobile devices in order to capture and use this information by interacting with physical objects. Finally, the goal of Physical Mobile Interaction is to exploit the interaction between mobile devices and physical objects in order to provide more natural and intuitive ways to interact with digital information and services that are associated with these objects. For example it would be more convenient and familiar to select a digital item by pointing at its associated object instead of searching for the same item in a glutted mobile application menu.
Different projects have developed solutions for Physical Mobile Interaction using different interaction techniques and technologies. The following examples represent a selection of the most common ones:

- Taking pictures of different visual markers that are attached to physical objects and the recognition of their codes is probably one of the earliest and most common Physical Mobile Interaction techniques [vico]. It is based on the work of [RoGf 04] and can be found in advertisements or on posters [RSH 04] where their selection initiates the invocation of associated context-aware mobile services or the retrieval of additional information. One of the most recent projects in this area, Semapedia [sema], provides the creation of visual markers that link physical objects with appropriate articles on Wikipedia [wiki] (Figure 2.12a).
- During December 2005, german mail-order firm “Quelle” put up advertisement posters that included infrared transmitters. People could activate the infrared-feature of their mobile phones, hold it against the transmitter and participate in a raffle for a shopping coupon by automatically sending and receiving text messages [heis].
- In 2005 two trials were set up in Germany and France in order to test and evaluate the use of NFC in combination with mobile phones, an approach that is similar to NTT DoCoMo’s established i-mode FeliCa service [feli]. In October 2005, 200 residents of Caen, France could use NFC-enabled mobile phones in order to pay in selected retail stores or to access car parks [phil a]. Earlier that year, in April, electronic ticketing based on NFC was tested in Hanau, Germany (Figure 2.12b) [phil b].

a) **b)**

Figure 2.12: Using visual markers (a, from [sema]) and NFC (b, from [phil b]]) in the real world

2.5.2 The Internet of Things

The development of Physical Mobile Interaction is advanced and partially overlapped by an increasing industrial effort to tag everyday objects with contactless RFID markers which facilitates their automatic identification, tracking, monitoring and recognition in manufacturing, transportation and logistics. That way, physical objects get individual digital identities, which also makes it easier to reference and present them on a network. This ultimately leads to an *Internet of Things* [Mel 03] in which objects as well as their associated information and services can be easily identified and accessed.

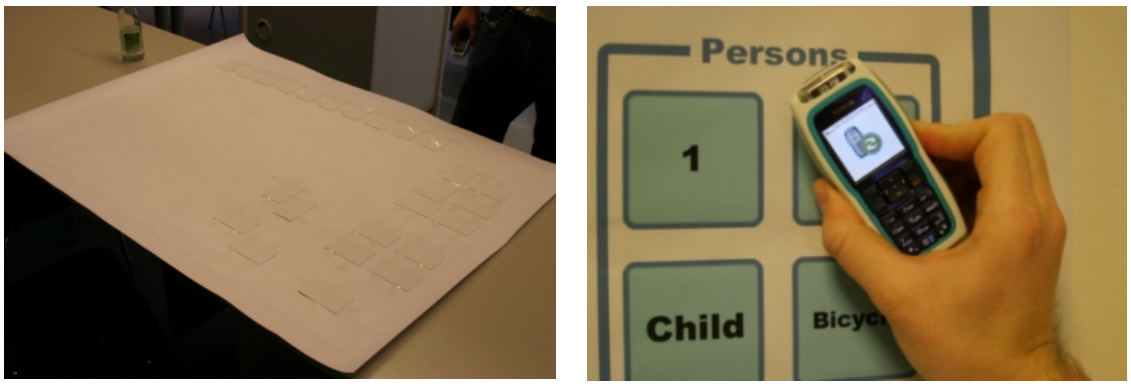
The infrastructure for weaving the Internet of Things is currently maintained by EPCglobal Inc. [epc] and is mainly based on 3 components apart from RFID-tags which are used to attach an unique *Electronic Product Code (EPC)* to objects. The *Object Naming Service (ONS)* is used to match the EPC of an object with the URL of its associated information that is stored on a server. The *Physical Markup Language (PML)* [pml] was developed to describe and store the information about an object on the network. PML descriptions of objects – whether provided by Web services or directly by NFC-tags - could be used as input for the invocation of associated Web services.

Driven by the industrial effort to automate and facilitate the transportation and logistics of everyday goods, or physical objects in general, the dissemination of Internet of Things-technology is growing rapidly. Physical Mobile Interaction could greatly benefit from this development and vice versa as the Internet of Things provides the technology that allows Physical Mobile Interaction to interact with its objects more intuitively and conveniently.

Chapter 3

Use Case Scenarios for Physical Mobile Interaction

In order to motivate and visualize the approach to more complex Physical Mobile Interaction, two use case scenarios were designed and developed that offer mobile ticketing through the interaction with augmented posters. These scenarios provide a foundation for rapid prototyping, testing and evaluating mobile interaction with physical objects but also make it more imaginable from the users' point of view.
For the use case scenarios, posters were augmented with Near Field Communication (NFC) [nfc] tags for Physical Mobile Interaction (Figure 3.1a). Similar to pushing buttons on an automat, users can touch different options and their values on the posters using a NFC-enabled mobile phone such as the Nokia 3220 with its NFC shell [nnfc]. The mobile phone selects options and their values through the recognition of the corresponding NFC-tags that are attached to the back of the posters (Figure 3.1b). Afterwards, a mobile client application can use the provided information for the invocation of associated Web Services. At this early stage, the NFC tags have only been attached to the posters in order to demonstrate the general interaction with mobile device. The tags become more important during the stage of implementation (see chapter 6).



a) **b)**

Figure 3.1: A poster with NFC-tags (a) and recognizing them with a Nokia 3220 (b)

These use case scenario posters are examples for interfaces that are distributed between mobile phones and physical objects: The interface of a mobile client application that is rendered from a Web Service interface description guides the interaction with markers on the physical object and assists in the invocation of the associated Web Service. The parameter-values for this invocation are stored on the NFC-tags and represent external options of the application's user interface. Instead of having to use long, glutted lists of options in nested menus, users can select these options more easily from the posters through Physical Mobile Interaction.
For the use case scenarios, 2 posters were designed that are intended to replace automats for buying movie and public transport tickets using Physical Mobile Interaction. The first poster allows the purchase of movie tickets and offers a number of appropriate options like movie title, cinema name, number of tickets/persons and timeslot, together with a selection of values e.g. 4 different movie titles or 4 different cinemas (Figure 3.2). In addition, the first design of the poster provides an option for buying transportation tickets in a very limited scope. This option was intended to demonstrate the interoperability of Physical Mobile Interaction for services across different posters – a feature that was not included in the final implementation of the framework and the prototype client application but which remains an issue of future work.

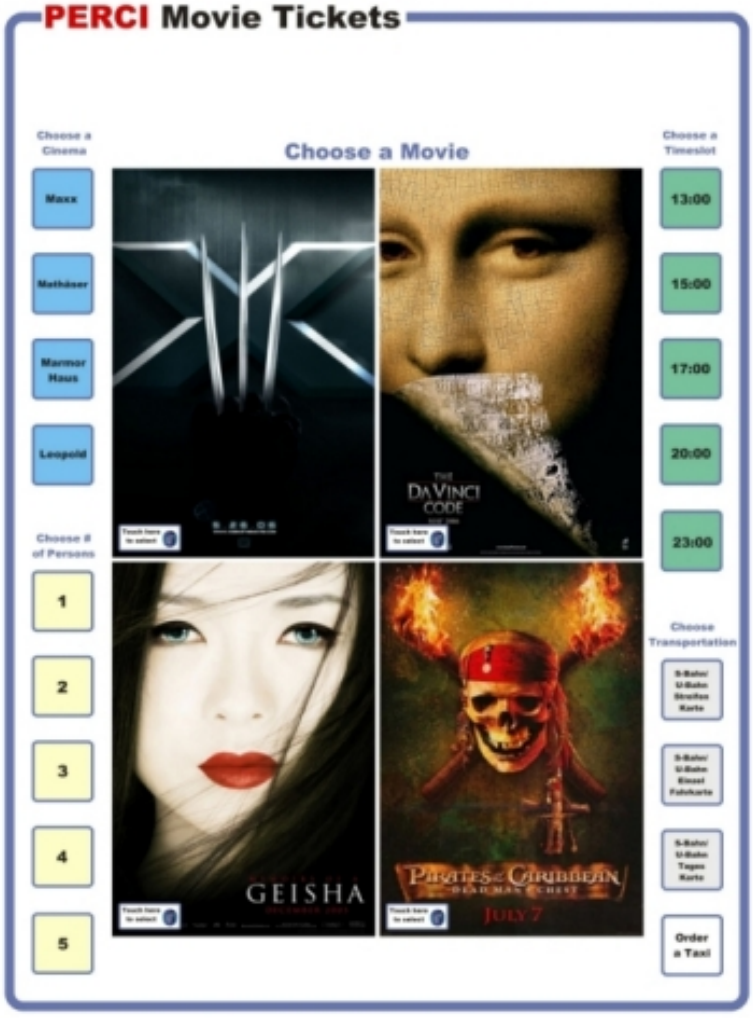


Figure 3.2: The movie ticketing poster [1][2][3][4]

[1] X-Men 3 Poster. www.actuacine.net/Poster/x-3.jpg. © 2005 Twentieth Century Fox
[2] The DaVinci Code Poster. www.actuacine.net/Poster/davinci.jpg. © Columbia Pictures
[3] Memoires of a Geisha Poster. www.actuacine.net/Poster/geisha.jpg. © Sony Pictures
[4] Pirates of the Caribbean, Dead Man's Chest Poster. www.actuacine.net/Poster/pirates22.jpg. © Disney Pictures

The second poster implements a simplified way to buy tickets for the public transportation system in Munich (Figure 3.3). Instead of having to understand the complicated ticketing system, inexperienced users only have to select the station they want to start their journey from, their destination, the number of persons and the duration of the ticket in order to have appropriate tickets suggested. Unfortunately there are too many stations too close together on the map to put a different tag behind each station. Since the pricing system of the Munich transportation system is based on 4 different areas (white, green, yellow and red), these areas were tagged instead. Users have to find their station of choice, remember its association to an area and touch the option, respectively the tag of the corresponding area. Another option on the transportation poster enlists the most frequently requested tickets. Users that are already familiar with the system can directly select their preferred ticket instead of having to assemble several parameters.
The last feature of the transportation poster is the so called *Activator Tag* that users have to select first in order to start the interaction with the other tags. The Activator Tag on the transportation poster starts this process explicitly, while touching any tag on the movie poster starts this process implicitly.

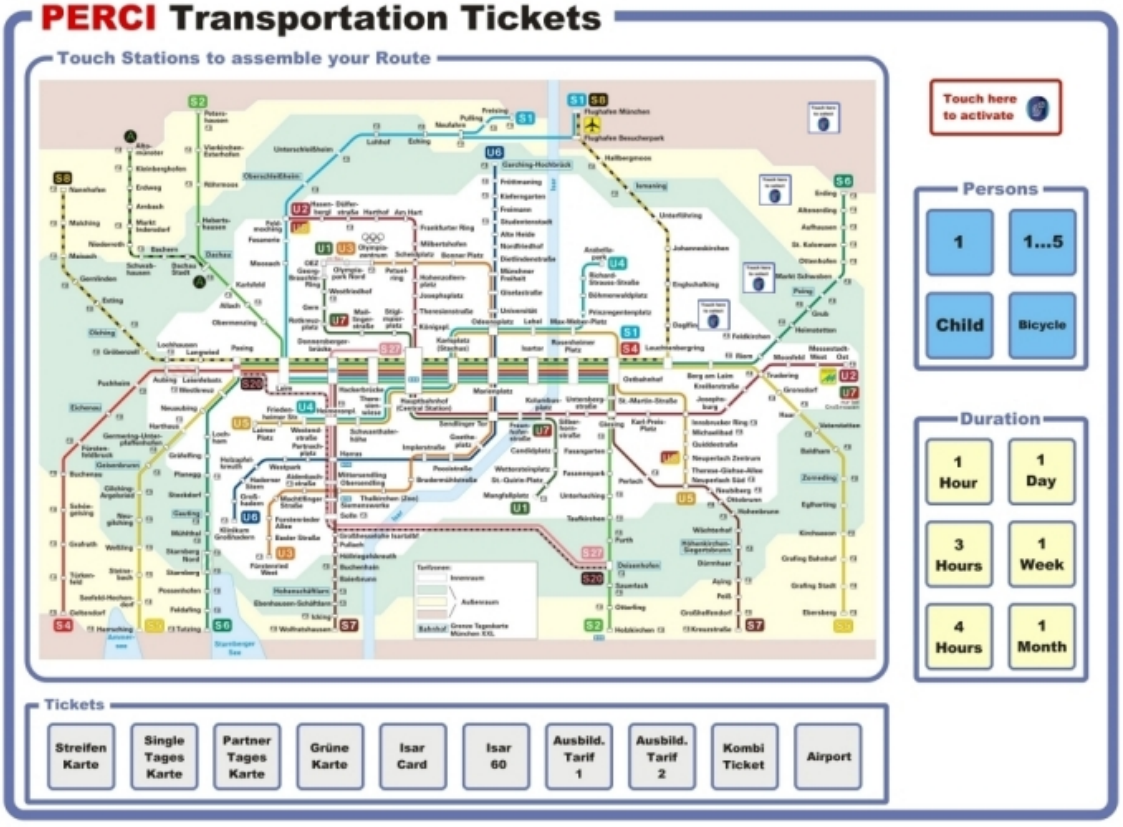


Figure 3.3: The public transportation ticketing poster [1]

[1] MVV Schnellbahn-Netzplan. http://www.mvg-mobil.de/pdf-dateien/netzplaene/schnellbahn netzplan.pdf

Chapter 4

Early Prototyping and Evaluation

Based on the two ticketing posters that were designed for the use case scenarios (see previous chapter), a first user study including some preliminary paper prototyping was conducted in order to evaluate mobile interaction with physical objects in general and the design and acceptance of the use case scenario approaches in particular. The results of this early evaluation provided valuable feedback for the improvement and refinement of the use case posters as well as for the implementation of the mobile client prototype application that followed.

4.1 User Study and Paper Prototyping Setup

In order to test and evaluate the chosen approach to mobile interaction with physical posters, two life-size versions of the use case posters were put up in a corridor of the Institute of Media Informatics in Munich. They were part of a user study that was conducted with 10 subjects, mostly students from the institute. All instructions, questionnaires and scenario descriptions that were used for this user study can be found in the annex.
Each subject of the user study was tested and evaluated in a separate session. At the beginning of each session, the investigator gave the subjects a short, general introduction about how NFC based interaction works and how they could interact with the posters and select their tags through a mobile device. The subjects could also use the Nokia 3220 and test the recognition of NFC tags on the posters themselves. After this introduction to Physical Mobile Interaction in the context of the use case posters, the subjects were asked to answer preliminary questions about how they liked the presented system, whether they would use it, which advantages and disadvantages it had and whether it was suitable for people without a technical background.
After these first questions, each subject was asked to accomplish two tasks – buying tickets for a movie and for public transportation – using the use case posters and paper-prototype mobile phones. These low fidelity paper-prototypes included a mock-up of a client application that subjects could use in order to virtually interact with the posters, receive feedback on their Physical Mobile Interaction with them and invoke the associated services (Figure 4.1). For this purpose, each paper-prototype showed a picture of a mobile phone and a mock-up of one of the different screens of the virtual client application. These screens modelled the flow of the virtual client application just like a real application would do. Subjects were told to use these paper-prototypes just like a normal mobile device and client application for interacting with the

posters. While they did so, the investigators exchanged the different prototypes and mock-ups as reactions to what subjects did or as instructions for what to do next.

Figure 4.1: Paper-prototype mobile phones and client application mock-ups

As a part of the scenario setup and the implementation of the final system, a server had to be contacted at the beginning of the interaction with a poster. This process modelled the generation of an initial interface from Web Service descriptions that would assist subjects with the further Physical Mobile Interaction. Although the two scenarios and their posters basically worked the same way, the initial interface of the movie poster should be generated upon touching a random option while the transportation poster required subjects to touch its Activator Tag first in order to start this process. This small difference was part of the interaction design in order to evaluate the usability of the Activator Tag as a definite starting point for the interaction. The approach of the movie poster comprised generating the initial interface implicitly which might cause an attention gap if the subjects had to wait for the interface although they expected an immediate feedback on the selection of the parameter option. The Activator Tag might anticipate this delay and make it expectable as it explicitly generates the interface.
Independent of starting the interaction implicitly or explicitly, the generated form-based interface could be filled with different values, e.g. a specific movie title or the destination of a journey, by touching the corresponding options on the posters. After all parameter-values had been assembled, the service that was intended to be associated with a poster would be called and generate the next interface, e.g. for payment details.
In order to keep the experiment short and concise, its focus lay on the Physical Mobile Interaction using the initial interface. The complexity of the rest of the service was reduced to a few simple steps. For the same reason, the experiment did not handle negative feedback, e.g. if no reservation could be created with the provided parameters. While the subjects were interacting with the posters and the paper-prototypes (Figure 4.2), the investigator was taking notes about remarkable behaviour, e.g. if a subject took extremely long to understand something or missed a step of the interaction.

After the subjects had completed the two tasks, they were again asked whether they would use the system if it was available, which advantages and disadvantages it had, what the general usability of the system was like, what could be improved, whether there were enough instructions and hints in order to operate the system, whether they considered the Activator Tag to be useful and which other scenarios they could imagine for the system.

Figure 4.2: Photos from the first user study

4.2 Results

The results of the user study and especially the questions that were asked before and after the use case scenarios, can be summarized as follows: 80 % of the subjects thought that the presented system was useful and would also use it if it was available – depending on the quality of the implementation and the number of available options. It was often mentioned that such a system could easily replace other automats (e. g. ticketing machines) but also that the human contact and individual feedback was lost compared to e.g. buying tickets at a counter. Although people without a technical background might at first have a certain inhibition level, they are considered to understand and successfully use the application – provided that they are familiar with using mobile phones. The system is largely considered to be intuitive and easy to use even when a certain initial effort is needed to understand it and to get accommodated to it.

When asked about advantages of the system the subjects answered that it is faster and cheaper than regular automats and would cause less queuing. The posters are inexpensive, can be put up everywhere and are easy to replace. Users are not bound to their desktop computer for ordering tickets online but can do it spontaneously using their mobile phones. The physical interaction is considered to generate fewer faults, to be easy and intuitive to use and to provide an additional value to the omnipresent mobile phone, especially in combination with mobile payment. The new application also causes technical sensation and interest, is considered to be visually interesting and attractive and provides a certain amount of fun. The distributed interaction between posters and mobile phones raises new possibilities for designing the interaction, pushing options from mobile phone menus onto posters and making them less complicated. The provision of a mobile service can generally improve the system, e.g. by calculating the best ticket option from a short list of parameters.

The mentioned disadvantages of the system can be related to the system in general and to the concrete application and its lacks of design: For some participants the system was not intuitive enough to use. The posters can only be operated with mobile phones (alternatives should be provided) and have to be put up and actualised. The use of NFC-technology is mostly unknown and would have to be established for common use and interaction. Frequently switching the focus of user attention between the poster and the mobile phone was also seen as a disadvantage.

Many users asked for additional information, e.g. an introduction the system, more feedback during the interaction or a context-aware help for explaining options and the posters' workflow. Users also wanted to know exactly what happened with their tickets after they bought them and how their possession was realized. Some users mentioned that a general handling of errors should be provided including the reversibility of actions, e.g. if users want to correct their selections.

As for the transportation poster, most users did not approve the employed method of choosing stations by associating them with the small tags of the different areas. At first most users intuitively touched the appropriate stations to select them. As no feedback was provided from the client, they looked for other means of selection and mostly found the options for the different areas of the transportation map. This definitely raised the need to find a new way for a more intuitive selection of stations.

Another interesting issue was the Activator Tag that was placed on the transportation poster and which prompted potential users to start interacting with the poster by touching it. It was intended to be an explicit starting-point for the interaction and most users understood this concept. Nevertheless only half of them appreciated its intention as an entry-point that enhances the structuring of the interaction and the feeling of controlling it. The other half of the users considered it to be useless and wanted to start the physical interaction implicitly by touching any option on the poster. Sometimes it was even not clear, what was activated by touching this tag, whether it was the poster itself, the mobile client application or the other tags.

Other possible scenarios that were suggested for the presented system included different automats, e.g. for cigarettes, buying train- or plane-tickets, calendars or combinations of ticketing and access control, e.g. for parking.

4.3 Redesign

The results of the user study provide a foundation for the redesign of the posters (Figures 4.3 and 4.4) and a refinement of the approach to Physical Mobile Interaction that is implemented with them. While some enhancements – such as the handling of errors, the reversibility of actions or more feedback from the mobile phone during the interaction - can only be included with the mobile client application, the redesign of the posters concentrates on 4 aspects:
The most prominent new feature of the posters is their augmentation with visual markers. As a part of an interaction re-design, the posters – and also the prototype implementation of the client application (see chapter 6) – will support the interaction techniques Pointing and Direct Input in addition to NFC-based Touching. Therefore each option on the two posters was augmented with a visual marker that encodes a unique numerical identifier. Hence options can be selected and identified through the recognition of the corresponding visual marker. Similarly, NFC-symbols were attached next to the visual codes in order to indicate the position of the corresponding NFC-tag that is attached to the back of the poster. For using Direct Input, users simply have to type the name, the value or a provided shortcut of an option. For more details about the concrete implementation of these techniques, please refer to chapter 6.3.

Figure 4.3: The revised movie ticketing poster

The other more prominent addition to the posters also concerns a refinement of the overall interaction design. Instead of having only one kind of options or tags, the system now differentiates between action-tags and parameter-tags. This decision was grounded on the idea of the Activator Tag which was not received very enthusiastically. In a redesign of its concept of starting the interaction with the posters explicitly, each action-tag on a poster represents a different service that is associated with the poster while parameter-tags represent the parameters that are necessary for invoking these services. The greatest advantage of differentiating tags in this way is the possibility to re-use a single poster for several different services that use the

same parameters. The movie ticketing poster now provides a service for buying movie tickets and a service that shows details of movies, e.g. actors or a summary of the story. The public transportation service offers a service for buying tickets by specifying a set of parameters – origin, destination, number of persons and duration – and a shortcut service that only needs the preferred ticket as input.
Another advantage that is inherited from the Activator Tag is the more concise segmentation of the interaction workflow. Users have to select an action-tag respectively a service explicitly before they can select parameter-tags. Between selecting the service and selecting the parameters, there is a natural attention gap that can be used for generating the initial interface for the service invocation.

Figure 4.4: The revised public transportation poster

Since many subjects of the user study requested more instructions and hints on how to use the poster or were often unsure about how to handle Physical Mobile Interaction with it, a manual was put at the top of the redesigned posters. A short text introduces Physical Mobile Interaction by telling users to touch the NFC-symbol on options with their NFC-enabled mobile phones or to take a picture of the corresponding visual marker. In addition, a list of bullet-points explains step by step how to start the mobile client application, select an action/service and provide parameters for service invocation.
The last important issue of redesign only concerns the public transportation poster. It features a slightly new way of selecting the stations for starting and ending a journey, since the solution from the first poster prototypes was inefficient. The new approach still works with the different areas of the transportation system, but now provides accordingly coloured tags next to the map of the transportation area, that can be mapped to the different areas more intuitively (see Figure 4.4).

Chapter 5

Architecture and Components

5.1 Architecture Overview

The approach to Physical Mobile Interaction, through the automatic generation of adapted interfaces from Web Service descriptions, is reflected in a schematic overview of the architecture of the Perci-framework (Figure 5.1). Its purpose is to comprise and integrate Semantic Web Services and Physical Mobile Interaction into a coherent system that generates adapted interfaces for mobile interaction with objects from the real world. In order to bridge the gap between these two separate domains, the architecture also includes an *Interaction Proxy* component that acts as a generic intermediate.

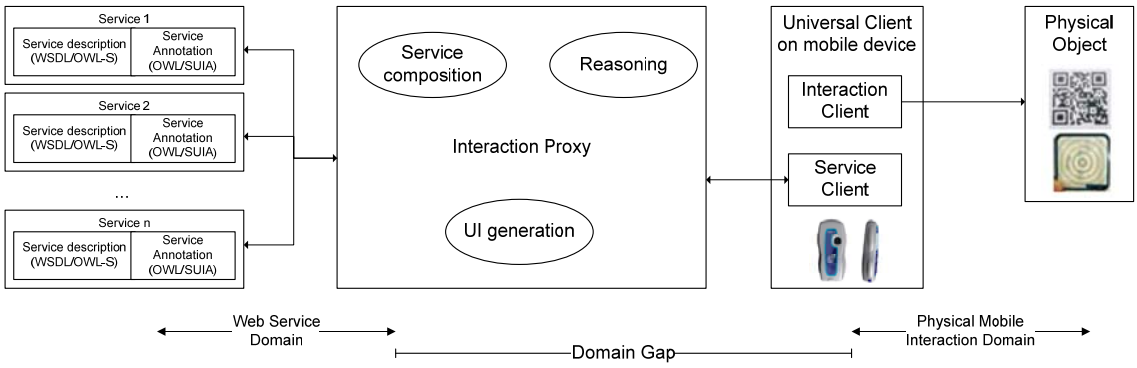


Figure 5.1: A high level overview of the Perci-framework architecture (from [Sior 06])

The *Physical Mobile Interaction Domain* of the architecture includes mobile devices and client applications that interact with physical objects. These objects are augmented with different technologies (e.g. NFC tags, visual markers, Bluetooth, ...) that provide information for the use and invocation of services that are again associated with the physical objects (e.g. a poster providing a screening schedule for movies that is used for ordering tickets via an associated Web Service).

These services can be found in the *Web Service Domain*. They are modelled as Semantic Web Services and represent the backend and service logic in the architecture. The descriptions of their functionalities, along with different extensions and annotations provide the basis for the generation of interfaces for Physical Mobile Interaction.
The Interaction Proxy mediates between Web Services and client applications. In order to increase the efficiency of the interface generation process, it adopts and centralizes common functionalities that are outsourced from Semantic Web Services (e.g. service composition or reasoning) and constrained mobile devices (e.g. resource-demanding transformation processes). That way the Interaction Proxy ties the two separate domains together while retaining their independence from each other and keeps the framework generic enough for the future integration and support of additional Web Services, client technologies and physical objects.
Dependent on the target platform of the mobile device, the Interaction Proxy assists the *Universal Client* application in the Physical Mobile Interaction Domain and provides complete interfaces (e.g. HTML) or compact interface descriptions for further rendering. The Universal Client itself acts as a mediator between Semantic Web Services and physical objects. It represents the generic application logic that renders, displays and uses these interfaces for the mobile interaction with physical objects and the invocation of associated Web Services.
The following chapters give a more thorough introduction to the Perci-framework architecture and its components. Since the focus of this thesis lies within the Physical Mobile Interaction Domain, they will provide as basic and profound an understanding of the architecture as necessary and serve as a prerequisite for the prototype implementation of a J2ME Universal Client in chapter 6. For more details about the framework architecture and especially its prototype implementation, please refer to Sven Siorpaes´s diploma thesis [Sior 06].

5.2 Web Service Domain

5.2.1 Overview of Service Descriptions

Just as service logic is the basis for a framework that exploits Physical Mobile Interaction for the use and invocation of Web Services, service and interface descriptions are the basis for interface generation. Hence our approach requires services that can be easily described and extended with further annotations in order to enable and support automatic derivation of interfaces and controlling their interaction flow.

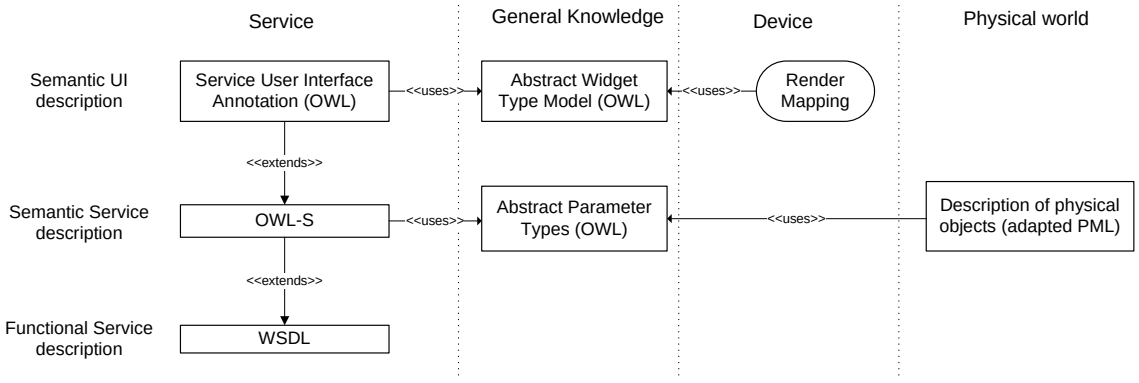


Figure 5.2: Overview of interface descriptions used in the framework architecture (from [Sior 06])

The services in the framework are largely independent but meet common standards in order to communicate and interoperate with the other framework components. Third party services should be able to become part of the framework through easy and minimal effort while retaining their independence instead of having to comply with proprietary and inflexible standards. In

order to meet these requirements we exploit and adapt the expressiveness of Semantic Web Services to model and describe services. This choice is reflected by the prototype implementation of the Web Service Domain. It was accomplished using the Apache Tomcat servlet container [tom], the Apache Axis Web Service framework [axis] and the Mindswap API [mind] for the automatic processing of OWL-S descriptions and the invocation of services.
Figure 5.2 shows an overview of the different service descriptions and extensions used within the framework as well as a classification of them. Although most descriptions are defined within the service domain, some of them are of general knowledge and hence independent from services, devices or physical interaction. They are extendable, reusable and essential in order to bridge the gap between Web Services and Physical Mobile Interaction as they describe common concepts that are related to each other and exist in both domains.

5.2.2 WSDL and OWL-S for Basic Service Description

The services attached to the framework are Web Services using the Web Services Description Language (WSDL) [CCMW 01]. WSDL allows extending and annotating existing Web Services and ensures interoperability between different applications and platforms by providing a standardised interface. It describes access to services and abstract message formats that are exchanged between services (e.g. names and types of the input and output parameters). Since WSDL service descriptions are not suitable for deriving interface descriptions from them, the framework additionally exploits the OWL-S [MBH+ 06] service ontology as a basis for the user interface generation process.
OWL-S builds upon WSDL by using it as a concrete grounding for invoking services. An OWL-S description defines a set of one or more atomic processes which are the building blocks of the service process model. Atomic processes have inputs and outputs which are mapped to the corresponding WSDL description. They can also be further composed to composite processes which define a specific control construct such as a sequence flow or a parallel flow. Exploiting this OWL-S feature of composing processes allows the modelling of different steps in interactions. For example a sequential interaction flow of two steps can be modelled as two atomic processes composed to a sequence control construct.
One of the main benefits of OWL-S is the assignment of self-defined *Abstract Parameter Types* which are required for the correlation between physical objects and service parameters (see Figure 5.2). Abstract Parameter Types are part of the common knowledge between physical objects and associated Web Services. This typing is crucial in order to relate information on a physical object to specific parameters of an associated service. Just as physical objects are associated with certain services (e.g. a movie poster for buying tickets via a ticketing service), information on these objects are related to input parameters of these services (e.g. options on the poster) and this association is expressed through the common use of Abstract Parameter Types. This correlation may be determined directly or inferred indirectly from the ontology topology. The description of corresponding information on the physical object is inspired by PML [pml], a markup language for specifying object properties that is used within the Internet of Things [Mel 03] (see chapter 5.3).

5.2.3 Service Description Extensions and Annotations

Since both WSDL and OWL-S are mainly abstract descriptions of how clients can invoke services, they don’t provide enough information for the description and finally the generation of user interfaces. Therefore the framework upgrades the standard WSDL and OWL-S service descriptions with additional extensions, that have to meet different requirements:

- An abstract type system for interface widgets which facilitates their rendering and the graphical representation of service parameters.
- Concrete type formats and constraints for service parameters that have to be met by the information provided through Physical Mobile Interaction (e.g. date formats).

- Predefined sets of valid values for an input parameter type.
- Readable labels to increase the expressiveness and usability of interfaces
- Additional descriptions that explain use and purpose of interfaces and widgets for the interaction with real world objects, e.g. getting a service parameter value by taking a picture of a visual marker.

In order to accomplish the description of user interfaces, the framework complements the standard WSDL and OWL-S service descriptions with the additional *Service User Interface Annotation (SUIA)* ontology that's also based on OWL-S (see Figure 5.2). The main class of the SUIA for annotating an OWL-S service description is the *AbstractUIMapModel*. It serves as a collection bag for several parameter mappings represented by instances of the class *ParameterMap*, which encapsulates all annotations for one service parameter. A single parameter mapping complements an input or output parameter in the OWL-S service description and indicates the required information (e.g. widget type, type format, label, value set, …) for automatically generating an user interface.
The value expressed by the property *hasAbstractWidgetType* is an *Abstract Widget Type* that the device rendering engine of the target platform has to interpret and map to concrete user interface widgets. Figure 5.3 shows the typology of the corresponding *Abstract Widget Type Model* with its different widget types that is specified in an external OWL model. It is also part of the general knowledge between the domains of Web Services and Physical Mobile Interaction and hence separated from them.

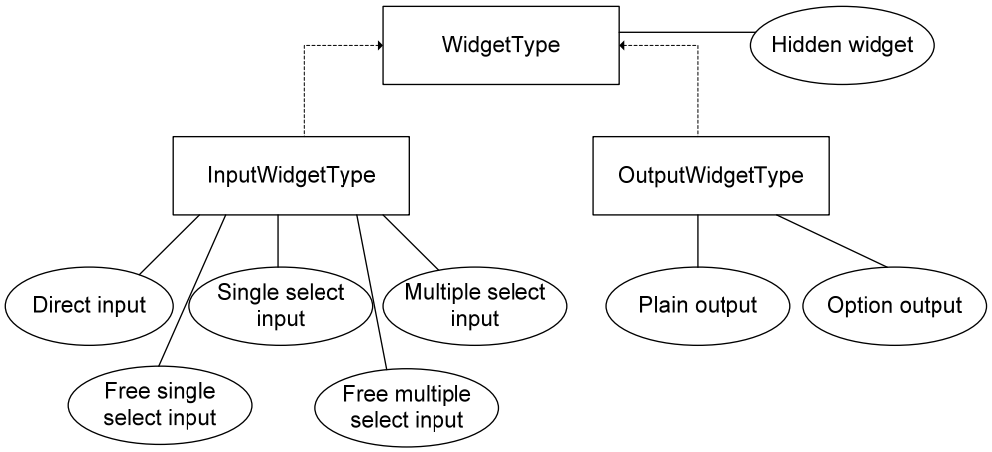


Figure 5.3: Overview of the Abstract Widget Type ontology used in the framework (from [Sior 06])

The hierarchy models the most common widget concepts in user interfaces and can be extended with additional components. Classes are represented as boxes whereas instances of classes are shown as circles. The general class *WidgetType* distinguishes between subclasses for inputs (*InputWidgetType*) and outputs (*OutputWidgetType*) depending on the type of service parameter to be augmented. Instances of inputs are:

- *Direct input* widget, which provides an arbitrary input, e.g. via a text field.
- A *single select input* widget provides a single value from a given choice, such as a drop down or radio button menu.
- A *multiple selection input* widget provides several values from a given choice, e.g. from a checkbox.
- *Single selection* and *multiple selection* widgets allow a loose interpretation of the widget, meaning that a widget may be rendered as direct input or selection input.
- A *plain output* parameter widget will usually be rendered as a simple textual message.
- An *option output* contains a set of options for an interaction step. For example services may determine a set of options that was not predefined but calculated during runtime.

- A *hidden parameter type* is a special instance of the general class *WidgetType* and should not be rendered by the target platform.

5.3 Interaction Proxy

5.3.1 Purpose, Scope and Implementation

The Interaction Proxy has been introduced into the framework architecture in order to mediate between the domains of Semantic Web Services and Physical Mobile Interaction. Its main purpose is to adopt or at least support the generation of interfaces from Web Service descriptions since implementing this process with either domain is not sufficiently efficient. The most prominent reasons are:

- Most mobile clients for which interfaces are to be generated are not powerful enough for handling this process as they are mostly constrained in terms of processing power, memory, storage capacity and the capacity of their network access.
- On the other hand, implementing the interface generation process in the Web Service Domain is inefficient as well, as each Web Service that is part of the framework would have to provide the whole mechanism for generating interfaces from its service descriptions by itself.
- Without an intermediate, mobile devices and client applications would have to call Web Services directly although many of them still lack the necessary APIs (such as the J2ME Web Service API JSR 172 [jwsa]).

Since implementing the interface generation with either the Web Service Domain or the Physical Mobile Interaction Domain is tedious, inefficient and inflexible, the Interaction Proxy acts as an intermediate between them. It adopts and encapsulates common functionalities from both domains. It takes the technological burden of interface generation from the Universal Client and inhibits the duplication of the process with each Web Service. That way the Interaction Proxy centralizes and manages processes that are related with each other instead of spreading them over the whole framework. Related components and common functionalities are tied together while the independence of specific Web Services, mobile client applications and physical objects from each other is reinforced. As a result, the framework is flexible and generic enough in order to easily integrate new, independent components with only slight customisation and guarantee its future extendibility with additional Web Services, client technologies and physical objects.

Although the framework architecture in Figure 5.1 models the Interaction Proxy as a separate standalone component on its own server, it can also be (even partially) implemented with the Universal Client, provided that its mobile device is powerful enough to host Interaction Proxy functionalities. For the prototype implementation of the framework, the Interaction Proxy was implemented as a Java servlet [jase] on a separate Apache Tomcat Server [tom]. The transformation of interface descriptions and part of the interface generation was implemented with Apache Cocoon [coc], a XML-based framework for *multichannel publishing*.

As Figure 5.1 shows, the Interaction Proxy is mainly intended for bundling the automatic generation of interfaces, the composition of different Web Services and Web Service reasoning in order to resolve possible indirect type-based correlations between physical objects and services. Other adopted functionalities are the communication between Web Services and client application, tracking sessions consisting of subsequent calls to Web Services and interaction steps, managing service logic or handling context information. Nevertheless the focus of the implemented framework so far lies on the interface generation process as well as the communication between Web Services and mobile client applications which will be discussed in the following chapters. Reasoning and Web Service composition will be subject to future extensions of the Interaction Proxy (see chapter 8).

5.3.2 Description Transformation and Interface Rendering

The basic task of the Interaction Proxy is the automatic generation of adapted interfaces for Physical Mobile Interaction. In order to be efficient, flexible and individual, this process has to balance 2 main requirements: The generation of interfaces from extended Web Service descriptions with the least effort and reuse of resources as well as the generic support of different target platforms, rendering technologies, interaction techniques, user preferences and other context information. Especially these properties of the Physical Mobile Interaction Domain determine or restrain each other and have great influence on the generation of interfaces that invoke the same service but look different from each other. For example the same interface description can be rendered differently depending on which interaction technologies are supported by the mobile client device. The first prototype of the Interaction Proxy implements and supports the generation of interfaces for HTML- and J2ME-clients on mobile phones. Their implementation will show the multitude of factors and possibilities that both influence this process and arise from it (see below and chapter 6).
Figure 5.4 shows an overview of the interface generation process within the Interaction Proxy. The first step is the transformation and reduction of Web Service descriptions into the *Abstract User Interface (UI) Description* that is the basis for the further generation of interfaces and their adaptation to client properties.

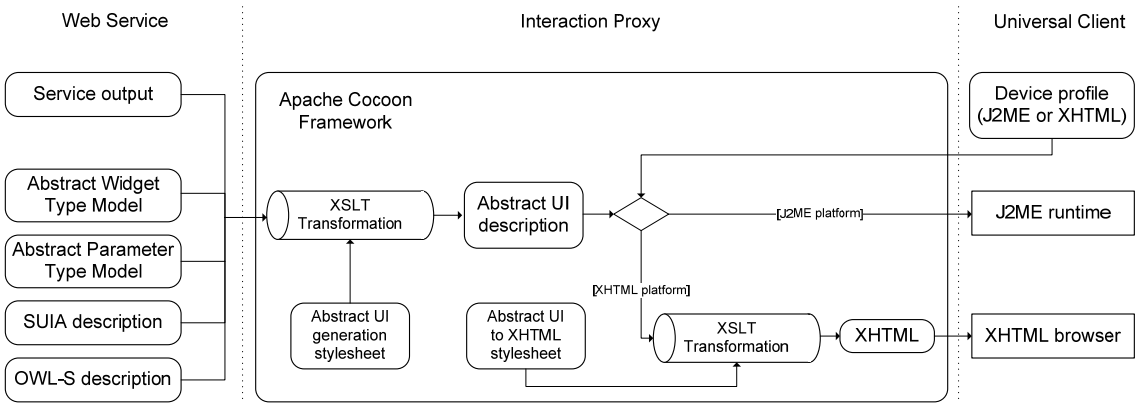


Figure 5.4: Overview of interface transformation and generation (from [Sior 06])

While the Interaction Proxy itself is implemented as a servlet, it uses the Apache Cocoon [coc] framework for the transformation of interface descriptions. Cocoon is an XML publishing framework that uses a pipeline-concept in order to generate content from XML-data. A Cocoon pipeline contains a number of proprietary components that generate SAX-events from different sources (e.g. input streams, files, directories, …), transform them using different stylesheets (e.g. XSLT, SQL, etc.) and serialize them into different target formats (e.g. XHTML, SVG, XML], WAP/WML etc.). Together with its built-in support for several different formats, Cocoon provides a number of mechanisms to control the generation and transformation of XML-documents which allows multichannel publishing. That way Cocoon can serialize the same generic XML-document into different specific output formats, depending on external data such as session- and request-parameters or other client-specific context information.
In order to generate an interface for the invocation of specific service methods, the Interaction Proxy gathers service descriptions, extensions, abstract user interface models and uses them as input for the transformation process (see Figure 5.4). Among them are the general OWL-S descriptions of the Web Service methods and the Service User Interface Annotations (SUIA, see Figure 5.2) that contain additional interface extensions. The Abstract Parameter Type Model and the Abstract Widget Type Model are taken from the general knowledge that is shared between the Web Service Domain and the Physical Mobile Interaction Domain (see Figure 5.2) and provide common definitions for service parameters and widgets. Additional input may be provided by the service implementation itself as the output of a previous interaction step - e.g. a

return value or a message on the outcome of a service method - may influence the next step.
The Interaction Proxy forwards these descriptions to Cocoon which aggregates them, applies the appropriate stylesheet and derives an Abstract UI Description according to its transformation rules (Figure 5.4). This new, abstract description can be seen as a summary of the previous descriptions that contains all necessary information for further interface generation while being more concise and easier to interpret.
Figure 5.5 shows an example of the Abstract UI Description: The root element *abstractUI* comprises a named group of different widget-elements; each of them will later be rendered into a separate widget of the final interface. Widget-elements have a mandatory *title*-attribute whose value is an instance of the *Abstract Widget Type Model* (see Figure 5.3) and determines the general type and purpose of the widget. The next steps in the interface generation will decide which specific interface widget will be rendered from this description. In the example below, a widget of type *singleSelectInputParameterType* could be rendered into a HTML text-input-element. Such mappings from widget type to interface element vary with different implementations and are also dependent on external factors. The value of the *abstractType*-element is an Abstract Parameter Type which defines the data that is captured with the corresponding widget as a certain input parameter of the associated Web Service. The other sub-elements of the widget-elements contain information which are necessary for rendering a concrete interface widget, such as a label, a declarative description of the widget's handling or purpose, the path to an additional image, the type of the parameter value and eventually a set of predefined values.

```
− <abstractUI>
  − <group title="Cinema Ticketing Service">
    − <widget type="http://perci.medien.ifi.lmu.de:8080/axis/ui/ParameterTypeModel.owl#singleSelectInputParameterType">
        <abstractType> http://perci.medien.ifi.lmu.de:8080/axis/domain/cinema/cinema.owl#Time </abstractType>
        <label>Timeslot</label>
        <desc> Please select the timeslot in this form or on the physical poster if available.</desc>
        <image> http://perci.medien.ifi.lmu.de:8080/axis/serviceDescription/extendedCinema/image4.jpg </image>
        <parameterValueType>http://www.w3.org/2001/XMLSchema#string</parameterValueType>
      − <parameterValueSet>
        − <option>
            <value>14:00</value>
            <label>14:00</label>
            <desc>N/A</desc>
          </option>
        + <option></option>
        </parameterValueSet>
      </widget>
    + <widget type="http://perci.medien.ifi.lmu.de:8080/axis/ui/ParameterTypeModel.owl#directInputParameterType"></widget>
    </group>
  </abstractUI>
```

Figure 5.5: Abstract User Interface Description

The next step after generating the Abstract UI Description is the rendering of a concrete user interface which can be implemented with either the Interaction Proxy or the Universal Client in the Physical Mobile Interaction Domain. The decision of where and how to render the interface is part of the Universal Client's interaction design that is determined by the supported platform, interaction techniques or personal preferences of its users.
The Interaction Proxy prototype implementation supports the generation of interfaces for clients that provide a mobile XHTML-browser or run J2ME midlets. In order to decide how to proceed in the interface generation process and where to render the interface, the Interaction Proxy takes context information into account that is passed with the first request for a specific Web Service interface. The current implementation of the Interaction Proxy recognizes information about the target platform from user agent headers (HTML browser) and Http-request properties (J2ME-clients) and has Cocoon use this information for choosing different branches of its multichannel publishing process (see Figure 5.4). If the system recognizes a HTML user agent header,

another transformation is applied to the generated Abstract UI Description. This time, another XSLT-stylesheet is used for the transformation in which the Abstract UI Description and its different widget-elements are translated into a HTML-document with input fields, checkboxes or drop-down-menus. This document is returned to the Universal Client, interpreted and displayed for interaction by its HTML-browser. In case the Universal Client is recognized to be supporting J2ME midlets, the Abstract UI Description itself is returned in order to be rendered by the J2ME runtime environment of the Universal Client application according to its own rules (see chapter 6).

5.3.3 Communication and Intermediation

Being the mediator between the Web Service Domain and the Physical Mobile Interaction Domain, the Interaction Proxy controls the communication between Universal Clients and the Web Services they want to use. Figure 5.6 shows a generic overview of the service invocation and the interaction between the components in the framework architecture in order to point out the chain of steps in the communication process and the role of the interface generation:

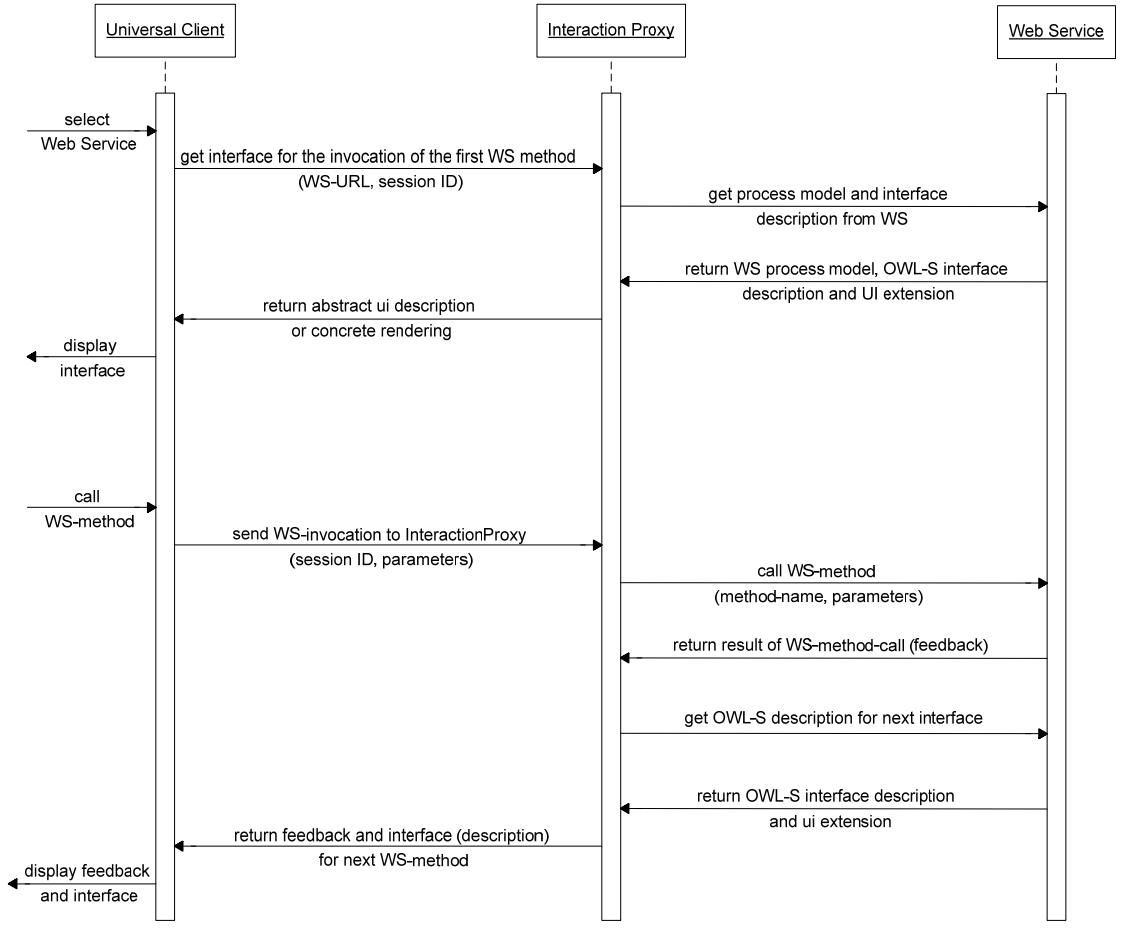


Figure 5.6: Generic communication between Universal Client, Interaction Proxy and Web Service Domain

For the first step of the service invocation, the Universal Client has to determine – preferably through Physical Mobile Interaction - the Web Service that is to be used. Before the service can be called, an interface for its invocation has to be generated. Therefore a request with the Web Service URL is forwarded to the Interaction Proxy which creates a session for the subsequent communication between the Universal Client and a Web Service. This session stores for

example user and client related context information, that can be used for the transformation of Web Service descriptions and the generation of interfaces (see above). The association of clients and sessions is handled via session-ids, whose provision depends on the platform and the implementation of the Universal Client.
The Interaction Proxy uses the service URL from the client request in order to acquire the process model of the corresponding Web Service and the interface description for its first method. The Interaction Proxy handles part of the service and interaction logic as it stores the process model in the client session in order to be able to forward subsequent request from the same Universal Client to the appropriate Web Service methods. Dependent on the client request, the acquired interface description is transformed and returned as either an Abstract UI Description or as a concrete interface description/rendering. Upon its reception, the Universal Client processes or renders the description and displays the corresponding interface.
The next client requests in the communication process are mere service invocations. The Universal Client collects values for service parameters through Physical Mobile Interaction and calls the Web Service that belongs to the current interface by sending a request with all service parameters to the Interaction Proxy. The Interaction Proxy recognizes the client through the provided session-id, forwards the parameters and their values to the appropriate Web Service method and receives a result or feedback from it.
The feedback of the service invocation represents a summary of its result and determines which interface description to return to the client. If the feedback is positive, the Interaction Proxy acquires the interface description for the next Web Service method in the process model and returns both feedback and interface description to the client. If the feedback is negative, it is returned together with the last interface description so that the client has to provide the same parameters again. This happens for example, if a ticketing service can’t provide a valid reservation for a certain order.
The following client requests for service invocations continue the same way until a service has finished. For the next Web Service invocation, the presented communication scheme starts again by choosing a Web Service and requesting the interface for invoking its first method.

5.4 Physical Mobile Interaction Domain

This domain comprises the devices, applications, technologies and objects that engage in Physical Mobile Interaction. The following chapters will explain the functionalities of the Universal Client, which technologies can be used for the augmentation of physical objects, how interaction with them can be accomplished and how it can be implemented.

5.4.1 The Universal Client

Just as the Interaction Proxy act as a mediator between the Web Service Domain and the Physical Mobile Interaction Domain, the Universal Client implements the intermediation between Web Services via the Interaction Proxy and the interaction with augmented physical objects. The Universal Client embodies the concept of a generic client platform and an application logic that integrates and uses the means of Physical Mobile Interaction for the invocation of Semantic Web Services and vice versa on an abstract level. It exploits Web Services in order to provide a functional background for more complex Physical Mobile Interaction and applies its techniques for easier invocation of the same Web Services. Although it integrates both domains, it keeps them and itself independent from one another.
The Universal Client can be used with different services without being proprietary or dependent on certain ones. It provides generic mobile interaction with physical objects for these services and supports different technologies and techniques for that purpose without being designated to a specific one. It supports different client platforms and can be implemented with different technologies.

As Figure 5.1 shows, the Universal Client manages two main logical components for its two most important tasks: The *Interaction Client* handles the mobile interaction with augmented physical objects and the *Service Client* supervises the communication with different Web Services via the connection to the Interaction Proxy. The realisation of these components as well as the use of certain technologies, services, interaction techniques or client platforms is always depending on concrete implementations of the Universal Client. So far the prototype implementation of the framework supports and implements Universal Clients that use HTML and J2ME.

5.4.2 Techniques and Technologies for Physical Mobile Interaction

In order to realize mobile interaction with physical objects, mobile devices and their Universal Clients can support different Physical Mobile Interaction techniques that are implemented with different technologies. Figure 5.7 gives an overview of common interaction techniques:

- *Touching* is an interaction technique where users select objects by touching them e.g. with their mobile phones. Most implementations of this technique are based on either Radio Frequency Identification (RFID) [Want 06] or Near Field Communication (NFC) [nfc] technology. Physical objects are augmented with NFC or RFID radio transmitter tags that store information, e.g. for identification or service invocation. Reading devices like the Nokia 3220 with its NFC shell [nnfc] (see Figure 5.7b) or the IDBlue RFID pen [idbl] (see Figure 5.7f) can retrieve this information and make it available for applications by simply touching those tags which automatically triggers the transfer of the stored information. This interaction technique is quite natural and allows unambiguous selection of objects due to the proximity between tag and reader that is necessary for transferring data.

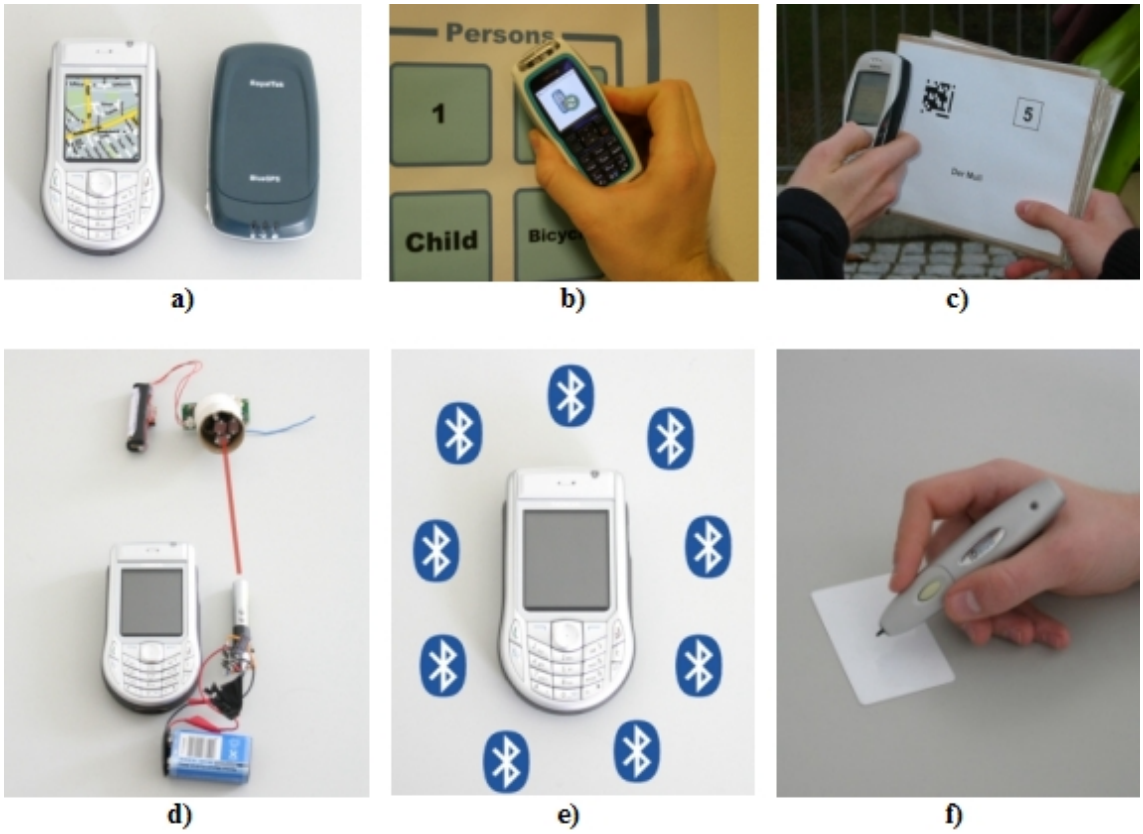


Figure 5.7: Techniques for Physical Mobile Interaction

- Similar to Touching, *Pointing* can be used to select objects in the vicinity of the user and to interact with them. A close-range implementation of this technique augments physical objects with visual markers that can be photographed with a mobile phone camera (see Figure 5.7c). Visual markers work like 2-dimensional barcodes and can be directly interpreted by a software algorithm on mobile phones [RoGf 04]. The amount of coded information is usually too small to be used for something other than the identification of a marked object. Thus it is usually sent to a server for the retrieval of additional information, as for example with the Semapedia project [sema].
  Another implementation of *touching* presented by [Lei 06] uses a laser pointer that is attached to a mobile phone (see Figure 5.7d). Physical objects are augmented with light sensors and micro-controllers with an integrated recognition algorithm in order to determine whether a light beam is pointed at them. If a sensor recognizes a light-beam, a message is sent to a pointing server which makes the identity of the selected object available via a Java servlet. In the current implementation, mobile client applications have to poll this servlet for the existence of such information in order to receive feedback on a selection.
- *Scanning* is another interaction technique used by [Lei 06]: It uses the built-in Bluetooth capabilities of mobile phones (see figure 5.7e) in order to search for, connect to and interact with other devices in a smart environment.
- Due to the dissemination of commercial solutions for mobile navigation, *Location Based Object Selection* can also be considered as another interaction technique. For this technique, mobile applications can request context information about their current position, e.g. from external GSM devices via Bluetooth (see Figure 5.7a). Similar to other location based applications, the proximity between user/device and objects can guide their selection, e.g. if a user moves within a certain section or range of an object.
- *Direct Input* is probably the most trivial yet most widespread interaction technique. Many mobile devices don't provide basic technologies for the implementation of the afore mentioned techniques or are not powerful enough to achieve required information processing. Direct Input simply uses the widgets of its client platform in order to provide an application with information that would otherwise be stored on wireless tags or coded into a visual marker (see Figure 5.7c). To be efficient, Direct Input information should be short or mapped to distinct shortcuts or mnemonics.

5.4.3 Client Platforms and Implementations

The actual implementation or use of the Universal Client is mostly depending on the client platform of mobile devices (e.g. Symbian, Windows Mobile, EPOC, etc.), supported technologies (e.g. Bluetooth, NFC, GPS, Camera, …) and programming languages (e.g. HTML, J2ME, etc.) as well as individual user preferences. The advantage of keeping the Universal Client as independent from platform technologies as possible can also turn out as a disadvantage if technologies that are necessary for implementing certain interaction techniques are not available. Therefore different implementations of the Universal Client for different platforms vary in the complexity of their functionalities and capabilities.
The prototype implementation of the framework currently supports the generation of interfaces for HTML and J2ME clients. The prototype implementations of the corresponding Universal Clients are presented below (HTML) and in chapter 6 (J2ME). These technologies have been chosen because they are the most widely used ones and because they show two different ends of the spectrum of supporting Physical Mobile Interaction techniques. While J2ME-clients can support practically every presented interaction technique, HTML-clients can only provide a form-based interface for Direct Input.
Other platforms and languages that seem to be suitable for implementing the Universal Client but which have not been considered in the present prototype implementation are Macromedia Flash Lite [mfli], Python for Nokia Series 60 mobile phones [ps60] or Opera Platform [oppla]. This last SDK is a framework that uses AJAX technology for the development of dynamic web

applications whose services interfaces - based on HTML, CSS and JavaScript – are stored locally on mobile phones. XML-encoded data that is transferred from a server only updates an interface, but does not generate an entirely new one, which reduces internet traffic.

Basic HTML Rendering

The most basic mobile client platform that is suitable for implementing an Universal Client is also the most restricted one. (X)HTML-browsers for mobile phones (e.g. Opera Mobile [opmo], Nokia Open Source S60 Browser [ChMo 06]) are powerful enough for rendering and displaying common web pages but their closed system and the static rendering of HTML-pages inhibits the integration and use of technologies for Physical Mobile Interaction. Even Java-based browsers such as Opera Mini [opmi] are not accessible or extendable from the outside.
Figure 5.8 shows several screenshots from a form-based HTML prototype interface for the Universal Client that is generated from the Abstract UI Description and returned to HTML-clients by the Interaction Proxy. All abstract widget descriptions have been rendered into appropriate HTML form elements and can be used to invoke associated Web Services with the URL of the Interaction Proxy as the value of the forms' action-attributes.

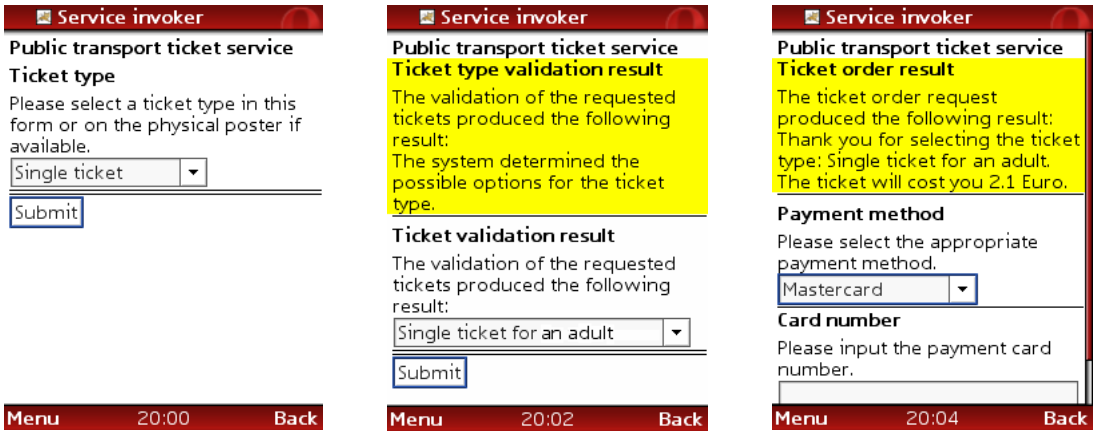


Figure 5.8: Screenshots from the HTML-implementation of the Universal Client (from [Sior 06])

Though being at the bottom end of the spectrum of possible Universal Client implementation, this HTML-prototype can be used with any device that has a HTML-browser, be it a mobile device or a desktop computer. Since Physical Mobile Interaction techniques and technologies can't be integrated with the browser, interaction with physical objects is restricted to Direct Input. This prototype implementation of a HTML Universal Client has been tested with an Opera Mobile browser on a Nokia 6630 mobile phone.

J2ME and PMIF

The second prototype of an Universal Client that is currently supported, was implemented with J2ME which allows a more flexible use of different Physical Mobile Interaction techniques. In order to gain easy access to the corresponding technologies, the prototype implementation which is presented in more detail in chapter 6, uses the *Physical Mobile Interaction Framework (PMIF)* [RWS 05].
The PMIF-framework has been designed for the easy and straightforward development and implementation of applications that use Physical Mobile Interaction. Its power and effectiveness is based on two aspects: On the one hand PMIF comprises several components that support different physical interaction techniques (e.g. Touching or Location Based Object Selection) and encapsulates the technologies behind them (e.g. Bluetooth, NFC, GPS). On the other hand, PMIF also steps back from specific techniques and technologies and provides a generic framework for uniform integration and easy use of these specialized components.

With PMIF it is easy to use different communication technologies for the interaction between mobile devices and physical objects. The framework supports several different technologies, yet provides enough abstraction to hide implementation details and to be extendable for new technologies. At the core of PMIF, the *interaction*-package and its generic classes (see Figure 5.9) support and facilitate Physical Mobile Interaction in a way that is independent of particular implementations and also transparent to the application and its developers.
The *InteractionManager* of the pmif.interaction-package is the central class for implementing mobile interactions with physical objects. It hides the complex actions for setting up and initialising generic Physical Mobile Interactions, manages their life-cycle and provides methods for assuring the support of different *InteractionTypes* (which present different interaction-technologies in the framework, like visual markers, Bluetooth or NFC).
Before starting the interaction with a physical object, an application has to register an *InteractionController* with the InteractionManager. Different InteractionController-objects set up connections with physical objects using the technologies behind different InteractionTypes. Applications never interact with InteractionControllers directly. They only register an appropriate InteractionController for the InteractionType they are interested in. The InteractionManager administrates the InteractionControllers and calls their methods in order to control the state of their life-cycle and to explicitly start physical interactions.
While registering an InteractionController with an InteractionManager only sets up physical interaction, applications eventually interact with objects that implement the *PhysicalObjectConnection*-interface. These are created by the different InteractionControllers as the result of successfully establishing a connection with a physical object. A PhysicalObjectConnection can be understood to provide a stream-metaphor which enables reading from and writing to physical objects using the technologies behind the different corresponding InteractionTypes. These allow the communication with objects, contain the data exchanged during the interaction and provide methods to retrieve it.

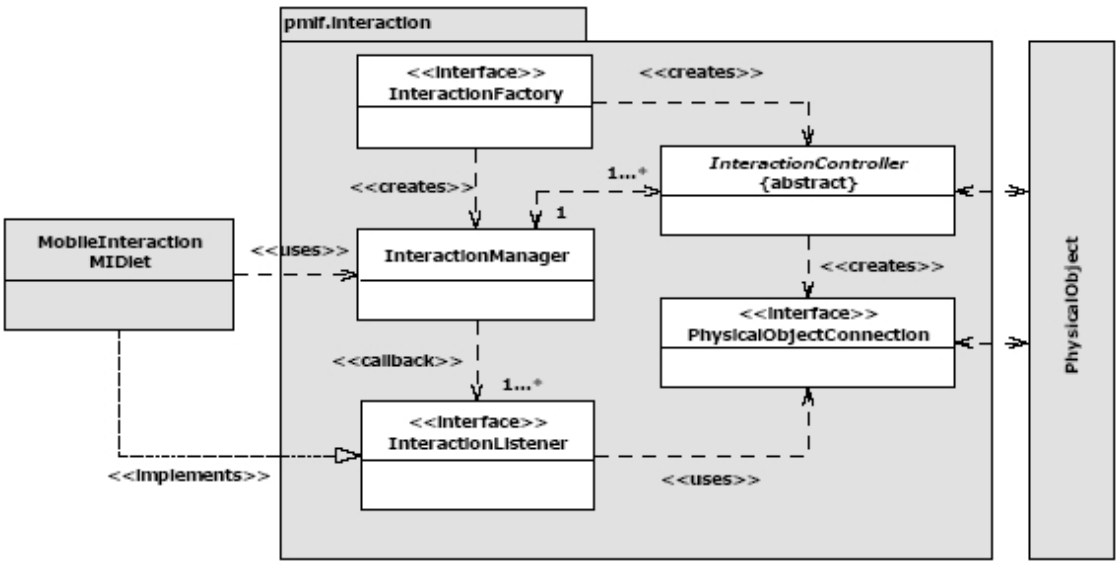


Figure 5.9: Main components of the generic PMIF framework architecture (from [RWS 05])

In order to use PhysicalObjectConnections, applications have to implement the InteractionListener-interface and register themselves with the InteractionManager for required InteractionTypes. As soon as an InteractionController that has been registered for a certain InteractionType has established a connection to a physical object, the created PhysicalObjectConnection is returned to the InteractionListener that has been registered for that same InteractionType. The object that implements this interface can now use the data from the acquired PhysicalObjectConnection in order to implement its own event handling and functionalities. The last component of this framework is the InteractionFactory-interface that

can be used for creating InteractionManagers and InteractionControllers as well as for their registration. While these standardized classes and interfaces enable abstract and uniform connection and interaction with physical objects, they also provide the foundation for specialized components that implement particular technologies for Physical Mobile Interactions. Due to the modular structure of PMIF only the InteractionController and the PhysicalObjectConnection have to be sub-classed and re-implemented in order to extend the framework for a new technology, e.g. NFC-based data-exchange.
In order to include PMIF interaction techniques with mobile applications and take advantage of the corresponding technologies, developers only have to implement a small number of standardized steps. The next chapter shows the implementation of these techniques as part of the Universal Client J2ME prototype implementation in more detail.

Chapter 6

J2ME Prototype Implementation

6.1 Generic Prototype Overview

The Universal Client is the generic platform that realizes the integration of functionalities from both Web Service Domain and Physical Mobile Interaction Domain for their mutual benefit. Due to the flexibility of the J2ME platform and its wide range of supported technologies, the J2ME prototype implementation of the Universal Client follows its concept more closely than the HTML prototype (see chapter 5.4.3). Likewise it has been designed and implemented in order to be used with the posters from the presented use case scenarios (see chapter 3) as well as with the Web Services and the Interaction Proxy from the framework introduced (see chapter 5). In addition, it supports the interaction techniques Touching, Pointing and Direct Input (see chapter 5.4.2).
This and the following chapters present the implementation of the Universal Client prototype for the Java 2 Micro Edition (J2ME) platform in gradually increasing levels of detail. After an overview of the application's flow, its interaction and service design as well as its technical requirements, selected parts of the prototype implementation will be presented on code level in order to make its main challenges, tasks and components comprehendible.

6.1.1 Generic Application Frame and Application Flow

Following the concept of the Universal Client, its J2ME prototype implementation is not restricted to or dependent on the use of specific services and interaction techniques. These can be exchanged with other ones as long as they are part of the framework and conform to its standards. This universality is also reflected in the application flow diagram in Figure 6.1. It shows the course of the application as well as its scaffolding and how it works on a generic level. Its interfaces are not generated from interface descriptions, but are integral parts of the application. Together with the application logic, they provide a generic frame for the execution of Web Services and Physical Mobile Interaction.
The application starts with the *Main Screen* (Figure 6.2a) which introduces and manages the main functionalities of the application. As the first step of the service-interaction, users have to

select one of the available services which are represented by action-tags on the posters. Likewise, service options alternatively parameter values are represented by parameter-tags.

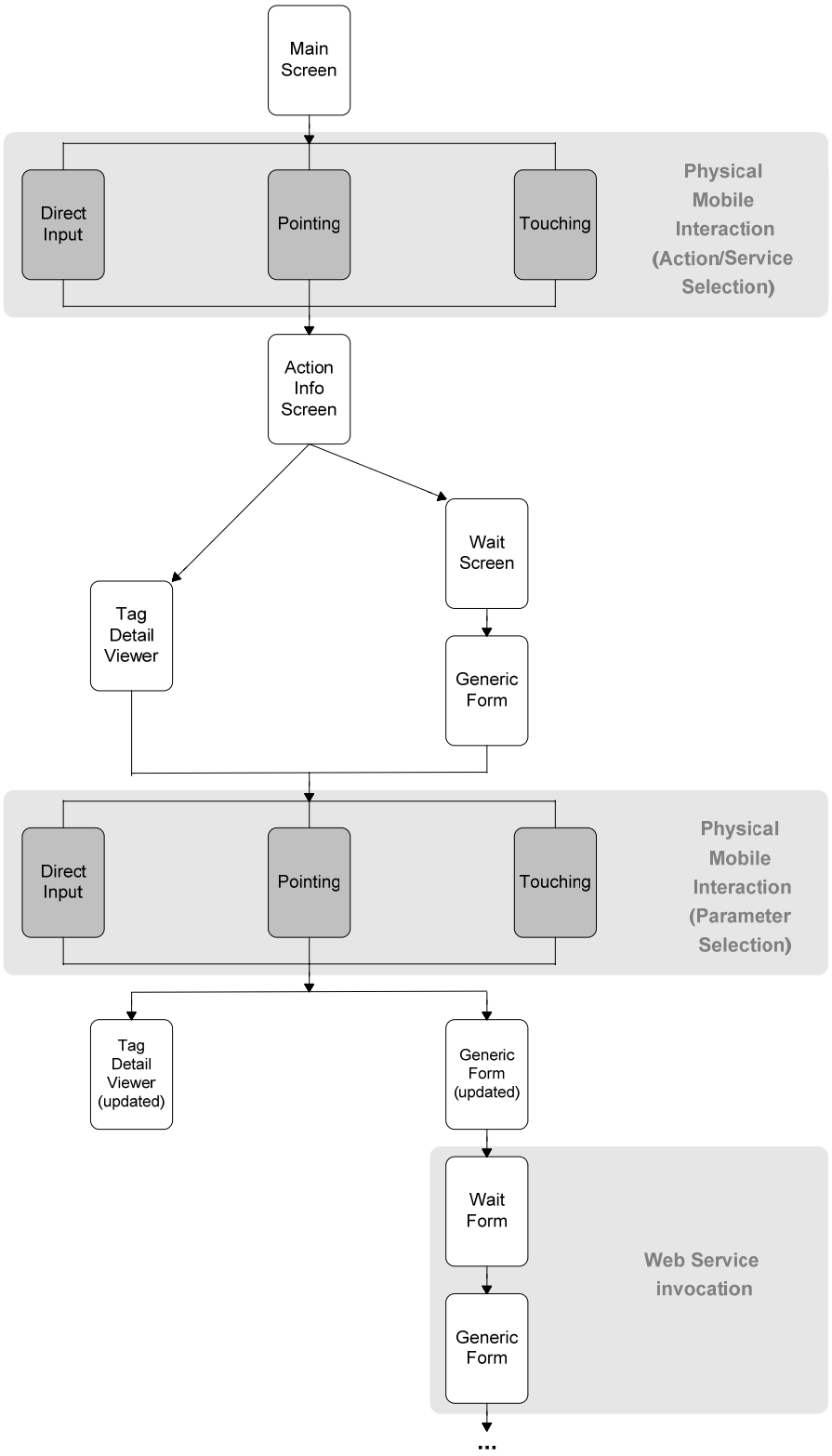


Figure 6.1: Generic application flow of the J2ME prototype implementation

The prototype implementation supports interaction through Touching, Pointing and Direct Input. Users can choose between these techniques in order to select the different tags and accordingly the services or options they want to use. The prototype assists users with this process and alerts them about false selections. In order to keep the application flow as easy to understand as possible, negative feedback is not depicted in Figure 6.1. The interaction design and the available services of the prototype implementation are discussed in more detail in the subsequent chapters.
After the user has selected a service, the application presents more information about it on the *Action Info Screen* (Figure 6.2b) which implements a transition to the actual service interface. Depending on the type of service, the application flow takes 2 different directions: The prototype includes a service for the movie ticketing poster called *Tag Detail Viewer* (Figure 6.2e) that provides additional information about selected movies, e.g. cast or director. The interface for this service is an integral part of the prototype implementation. All other service interfaces that are not included with the application are dynamically generated from Web Services interface descriptions (see chapter 5.3.2). Users can trigger this process from the Action Info Screen and the application shows a *Waiting Screen* (Figure 6.2c) which indicates the connection to the Interaction Proxy and the Web Service.

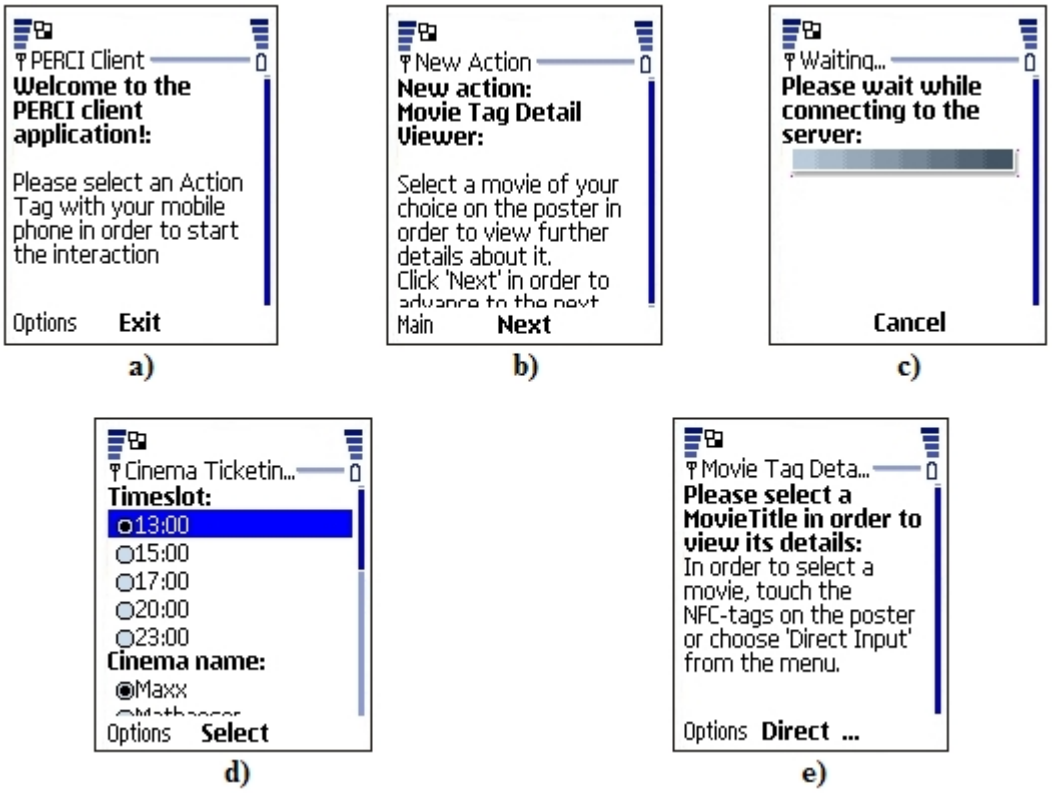


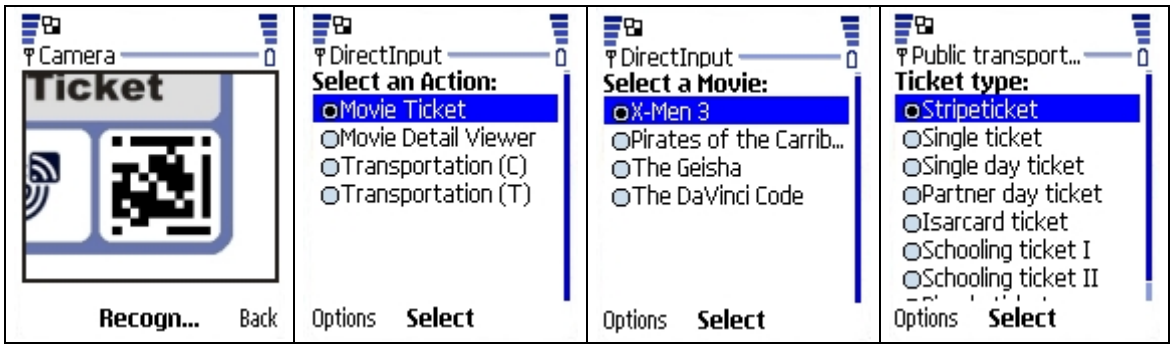
Figure 6.2: Examples of generic interfaces from the prototype implementation

After the application has received the interface description, it generates and renders the interface for the corresponding Web Service using a *Generic Form* component which can be understood as a generic frame for the generated interfaces (Figure 6.2d). In the next step users select parameter values and accordingly options on the poster in order to fill the interface, using the same interaction techniques as for the selection of the service. Whenever a user selects an option form the poster, the application updates the interface with the corresponding value.
After all parameters, that are necessary for the invocation of the current service have been collected, the user can submit them to the Interaction Proxy. Again, a Waiting Screen is displayed and after the response from the Interaction Proxy, the prototype application shows the feedback and the interface for the next service method that have both been generated from the

returned descriptions (see chapter 5.3.3). In case the feedback from the service is positive, the interface for the next service method will be delivered. Should the feedback be negative because e.g. no reservation for a certain movie could be found, the old interface is displayed again in order to submit different parameters. This scheme is repeated until the user has provided all parameters for the different steps of a service invocation and has received a final result, e.g. a receipt for a ticket ordering.
After this generic overview of how the application works in general, the next chapters provide more details about the different Physical Mobile Interaction techniques and the services that are supported by the current prototype implementation.

6.1.2 Interaction Design

The J2ME prototype implementation supports the Physical Mobile Interaction techniques Touching (through NFC), Pointing (through the recognition of visual markers) and Direct Input in order to select options from the posters of the use case scenarios (see chapter 3). In the application, these techniques are coequal, mutually exchangeable and users can smoothly alternate between them during the interaction process. That way, users can start selecting the parameters for a service using Pointing, but can switch to Direct Input during the process, e.g. because the visual markers on the physical object have been polluted and are not recognizable. The provided interaction techniques are part of the generic prototype implementation but can be substituted with other ones.
Touching is the most intuitive technique and does not require a separate interface for its employment. NFC interaction is indirectly started together with the application and can be applied any time by simply touching a tag with the mobile device. The current interface of the prototype application is updated accordingly.
The interface for Pointing can be accessed from the Main Screen, the interface for the Tag Detail Viewer and the Generic Form for all other services. In order to take a picture of a visual marker, the prototype application displays a camera screen for taking pictures (Figure 6.3a). After photographing and recognizing a marker, the application returns to the previous interface that is updated with the newly acquired option.



a) **b)** **c)** **d)**

Figure 6.3: Interfaces for the interaction techniques Pointing and Direct Input

Direct Input allows the direct manipulation of (generated) interface widgets in the prototype application. The interfaces for directly selecting services/actions (Figure 6.3b) and movies for the Tag Detail Viewer (Figure 6.3c) are integral parts of the generic prototype implementation. Their options have been implemented as exclusive choices for easy testing during the development process but can be substituted with widgets for e.g. free textual input. Direct Input is automatically included with all interfaces that are generated and rendered from interface descriptions (Figure 6.3d). Users can simply manipulate the widgets of the interface.
The prototype implementation also uses various alerts and notifications to control the application flow and to inform users about the results of their interactions. The signalling effects

of these alerts are reinforced by additional feedback from the vibration alert and a flashing backlight. An example is the alert that is triggered if a user selects a parameter-tag before he has selected an action-tag (Figure 6.4a). This ordering of action-tag first, parameter-tag next is convenient in order to associate services and parameters in a logical way. Apart from that, notification alerts are displayed whenever a user selects an action (Figure 6.4b) or a parameter (Figure 6.4c) in which case the name of the parameter additionally confirms the selection.
During the course of the application, other alerts can notify users about false or missing inputs. An example is shown in Figure 6.4d which is displayed in case a user forgot to provide enough parameters for the invocation of a service. Other alerts can be used to check the existence and validity of free textual input or whether a parameter that has been selected, can be used with the current action. That way, the Tag Detail Viewer can for example only show details of selected movies, but not e.g. cinemas (see Figure 6.5c).

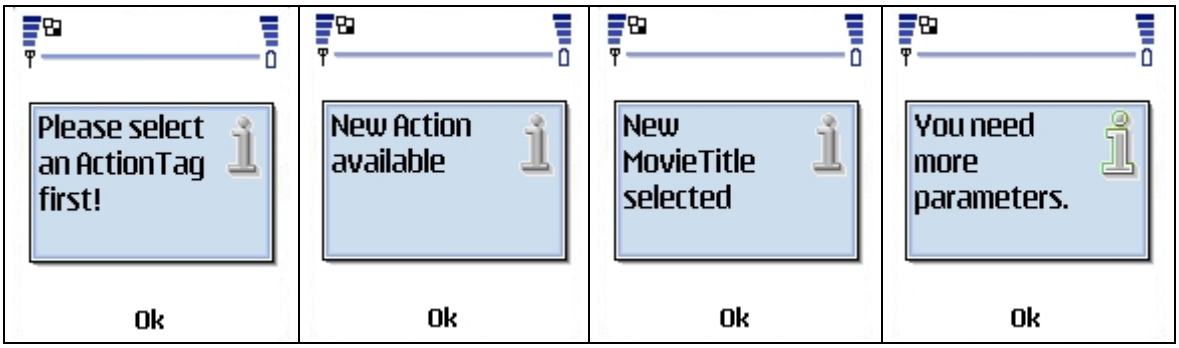


Figure 6.4: Alerts from the prototype implementation

6.1.3 Service Design

The prototype implementation of the framework includes 4 services that are associated with the posters from the use case scenarios. They can be understood as the content that fills the generic frame of the J2ME client application (see chapter 6.1.1). They provide the backend application and the J2ME prototype generates interfaces for their invocation from their Abstract UI Descriptions. Except for the Tag Detail Viewer which is part of the Universal Client itself, all services are in the Web Service Domain and have been implemented by Sven Siorpaes. Please see his diploma thesis for further details. Below, all 4 services and their different steps are introduced as far as their interface and interaction design is concerned:

(Movie) Tag Detail Viewer
The Tag Detail Viewer is part of the prototype application and can be used to show details about options on different posters. Its simple interface is an integral part of the client application and is not rendered from a Web Service description. The current application prototype only supports movie-tags on the movie poster (see chapter 3), from where this service can be selected.
After a user has selected the corresponding action-tag on the poster, a description of the Movie Tag Detail Viewer and details about its use are shown on an Action Info Screen (Figure 6.5a). From there, the user reaches the actual interface (Figure 6.5b) and can use one of the supported Physical Mobile Interaction techniques in order to select a movie. In case of selecting another option, an alert reminds users that this service only supports movie-tags (Figure 6.5c). After selecting such a tag, its description is displayed (Figure 6.5d). This information can be stored on NFC-tags, provided by the client application or retrieved from a server (see chapter 6.3).

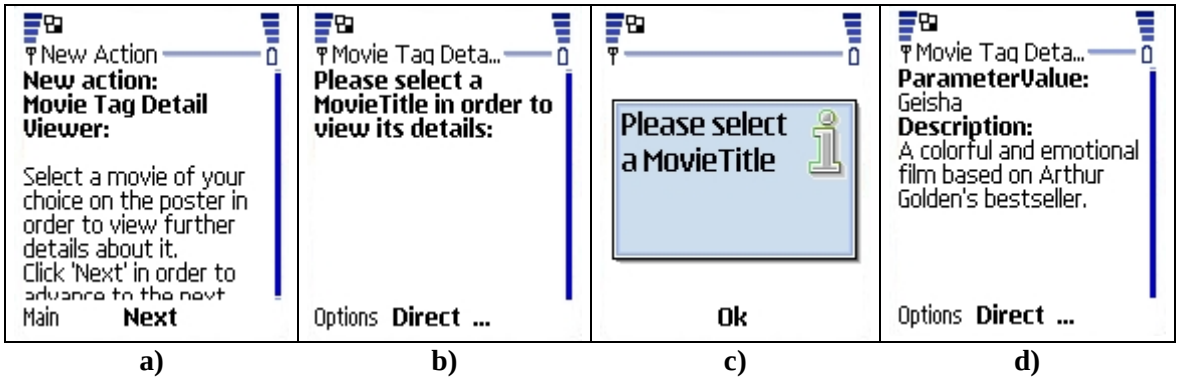


Figure 6.5: Screenshots from the (Movie) Tag Detail Viewer

Movie Ticketing Service

The second service from the movie poster can be used to buy movie tickets. After the selection of the corresponding action-tag, the service is again introduced on an Action Info Screen from where the user can initiate a request to the Web Service in order to retrieve the description for the first interface (Figure 6.6a, b, c). The generated interface asks users to provide their preferred timeslot, cinema, movie title and number of persons. Values for these parameters can be selected from the poster using the supported Physical Mobile Interaction techniques. After sending the request to the Web Service, a feedback on the outcome of the service invocation and the description for the next interface are returned. Both feedback and description are rendered and displayed as the next interface (Figure 6.6d, e). Provided that the feedback is positive, users have to give their name in order to make a reservation.

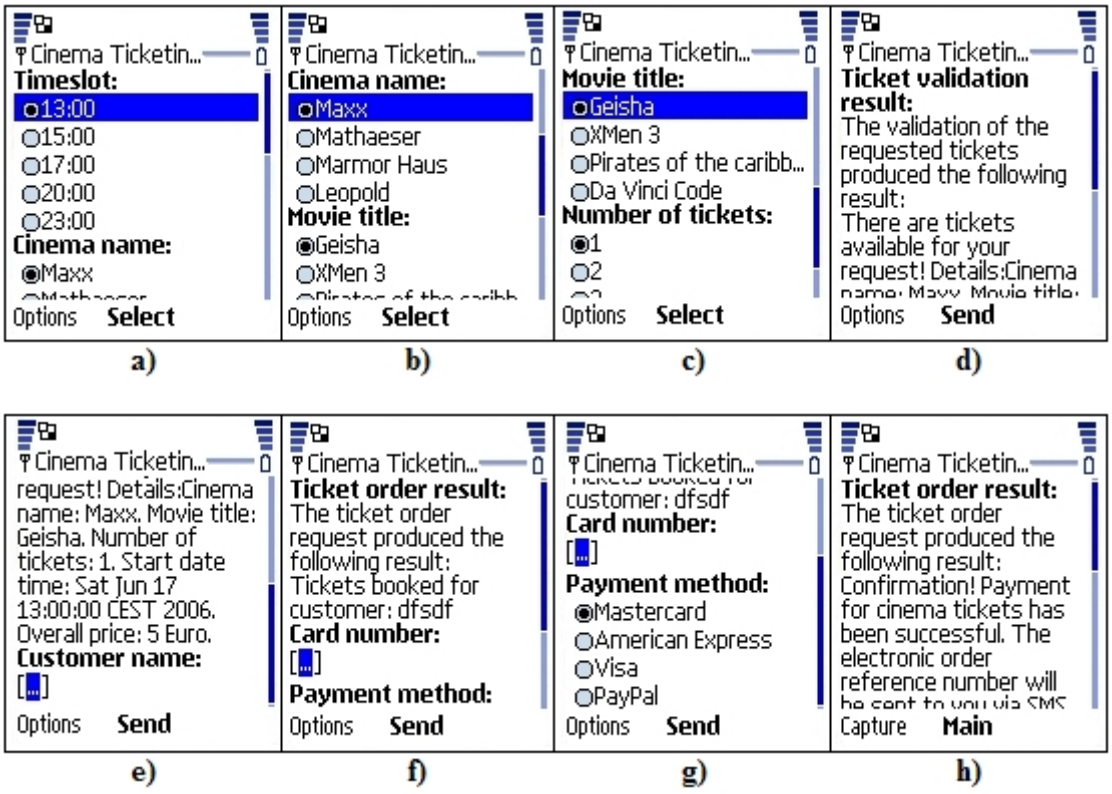


Figure 6.6: Screenshots from the Movie Ticketing Service

After submitting this request, the last step of the service asks for payment details like credit card and credit card number (Figure 6.6f, g, h). Afterwards, the service interaction is complete and users get a notification about the successful reservation. All these steps imply that the feedback is always positive. In case of a negative feedback, the description for the last interface is returned, so that users can enter new values for a valid service request.

Transportation Ticketing Service (Context and Type)
The two services that are associated with the transportation poster are very similar and provide two logically separated views on the same ticketing service. The first service called C*ontext* allows users to select the number of passengers, the duration of their journey as well as its origin and destination as parameters from the transportation poster. This is done using the first interface from the corresponding Web Service (Figure 6.7a, b, c), which has been generated the same way as the first interface of the movie ticketing service – selection of action/service tag, details on the service and initiation of the interface description retrieval and rendering. For the next step in the interaction flow, the service returns a list of recommended tickets that suit the provided request (Figure 6.7d, e). After the user has selected a ticket, the last step asks for payment details (Figure 6.7f, g) similar to the movie ticketing service. The last feedback is a notification about the success or failure of the transaction (Figure 6.7h).

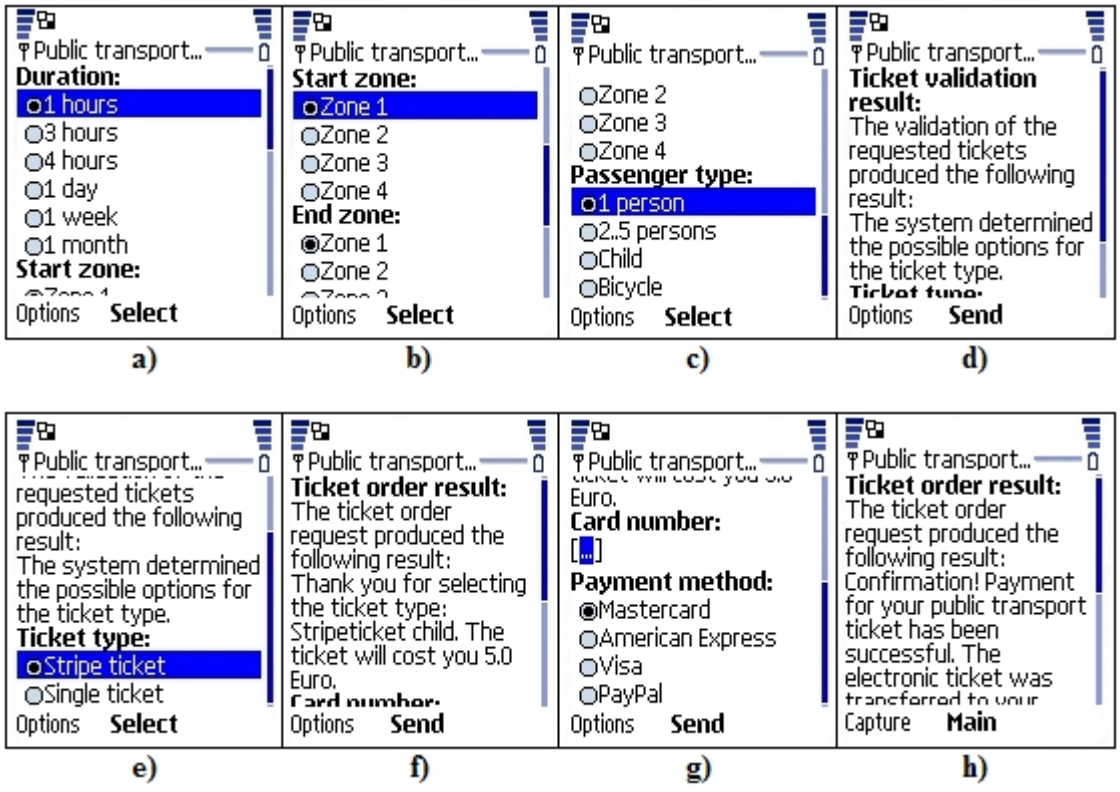


Figure 6.7: Screenshots from the Transportation Ticketing Service (Context)

The second transportation ticketing service called *Type* is some kind of a shortcut for the first one. While *Context* allows users that are unfamiliar with the transportation system to select their tickets by providing common service parameters, Type allows more experienced users to select their preferred ticket directly without a ticket recommendation from the system. The service starts similar to the previous one but the first interface contains a list of all available tickets (Figure 6.8a), which can be selected through Physical Mobile Interaction. The following steps for the type service are the same as for the *Context* service: refinement of the ticket order (Figure 6.8b, c), details of payment (Figure 6.8d, e) and a final notification about the result of the order (Figure 6.8f).

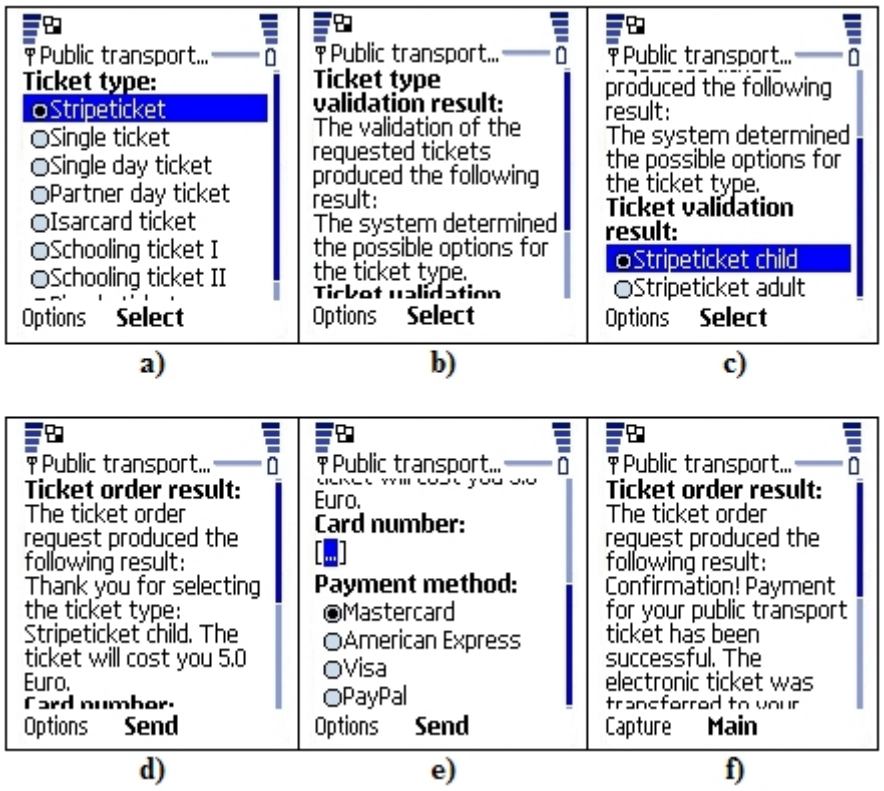


Figure 6.8: Screenshots from the Transportation Ticketing Service (Type)

6.2 Implementation and Technology

6.2.1 Hardware and Software

The prototype application implements the Physical Mobile Interaction techniques Touching and Pointing using NFC and visual marker recognition technology (see chapter 5.4.2). Therefore, it was predominantly tested and used with the Nokia 3220 mobile phone (Figure 6.9a) and its attached NFC shell (Figure 6.9b) [nnfc], whose reading range is approximately 3 centimetres. Although the Nokia 3220 is one of a very few mobile phones that support NFC features, it does not provide the necessary JAVA APIs for the recognition of visual markers. Hence the Nokia 6630 mobile phone (Figure 6.9c) is used for testing the Pointing interaction technique. Direct Input is supported by both mobile phones.

In order to interact with physical objects through Pointing and Touching, each option on the posters from the use case scenarios (ref) was augmented with a Mifare NFC tag (Figure 6.9d) and a visual marker (Figure 6.9e), so that they could be selected using the mobile phones and the prototype application. The visual markers were created with the visual code generation tool from [vico].

The implementation of the J2ME prototype application is based on the Java Platform Micro Edition platform [j2me]. It uses the Connected Limited Device Configuration (CLDC) 1.0 and 1.1 [cldc] as well as the Mobile Information Device Profile (MIDP) 2.0 [midp] from the Sun Java Wireless Toolkit 2.3 Beta [swtk]. In addition, the Eclipse IDE 3.1 [ecl] and the Nokia Prototype SDK 4.0 [npro] were used in order to code and test the implementation. The parsing of different XML-documents was implemented using kXML 2 [kxml].

The interaction techniques Touching and Pointing are implemented with the Physical Mobile Interaction Framework (PMIF) [RWS 05] (see chapter 5.4.2) which includes and supports the

corresponding APIs and technologies. Access to the NFC features of the Nokia 3220's NFC shell has been implemented with the Nokia NFC & RFID SDK 1.0 which is part of Nokia's Field Force Solution [nffs]. The recognition of visual markers is based on the work of Beat Gfeller and Michael Rohs ([RoGf 04], [vico]). Direct input has not been implemented with PMIF, since its support for this interaction technique is too inflexible.

Figure 6.9: Mobile phones and marker technology to support Physical Mobile Interaction

6.2.2 Software Architecture

Figure 6.10 shows a general overview of the prototype's software architecture with its main classes and their interactions in order to introduce the most important features and concepts of the client application. This diagram also serves as an outline for the following chapters - 6.3 to 6.5 - on the implementation and its details.

The main class of the client application is the *PerciClientMidlet* which manages common objects and important features of the application logic - especially concerning the interaction between physical objects and generated interfaces. Following the concept of the Universal Client, it creates and manages an Interaction Client and a Service Client (see chapter 5.4.1). The *InteractionClient* class encapsulates and handles the Physical Mobile Interaction techniques that have been implemented with PMIF and the *ServiceClient* class handles the communication with Web Services via the Interaction Proxy.

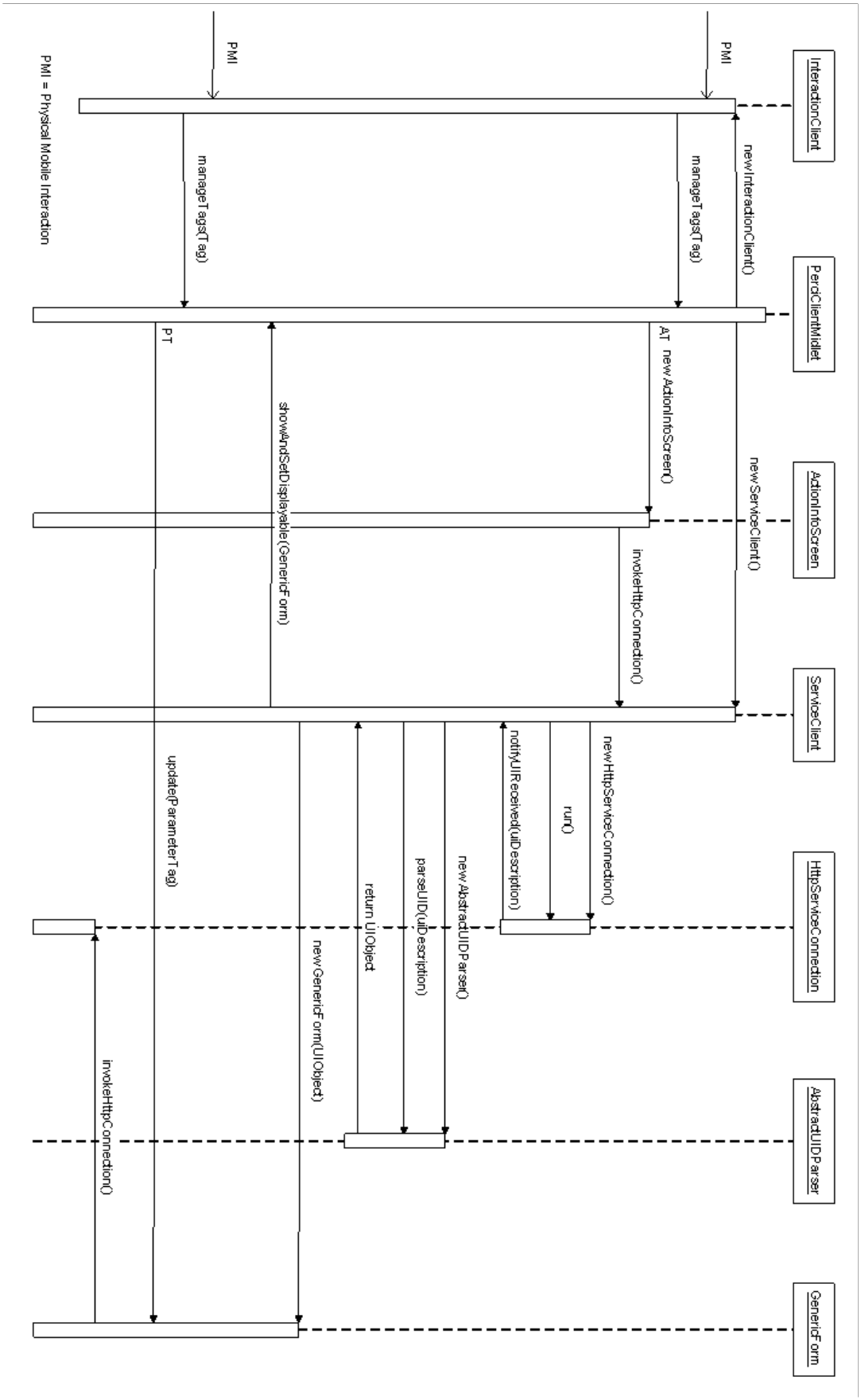


Figure 6.10: The software architecture of the mobile client application

Figure 6.10 shows how these and other classes interact with each other in order to realize the use of Physical Mobile Interaction for the invocation of Web Services: During the operation of the client application, the InteractionClient waits for Physical Mobile Interaction events, e.g. when a NFC-tag on a poster is touched and its data is retrieved. This data is encapsulated in an object of the *Tag* class, which is forwarded to the PerciClientMidlet via its *mangageTags()*-method. This method is a central part of the application logic. It acts as an interface to the Physical Mobile Interaction Domain and has to be used by all its mechanisms in order to have interaction events generated through Pointing, Touching or Direct Input processed. This method decides what to do with information gathered through Physical Mobile Interaction.
The first step in the interaction with a Web Service is the selection of its action-tag on a poster. After the mangageTags()-method has received such a tag, it creates and displays an *ActionInfoScreen* object in order to provide the user with more information about the selected action. From this screen, the user can initiate a connection to the Interaction Proxy in order to get the description for the first interface of the service - or to get to the local interface of the Tag Detail viewer (not depicted in Figure 6.10). In the first case, the ActionInfoScreen calls the *invokeHttpConnection()*-method of the ServiceClient object which has previously been created by the PerciClientMidlet. This method takes different parameters from the ActionInfoScreen, creates and runs an *HttpServiceConnection* object which handles the communication with the InteractionProxy in an new thread.
The answer from the InteractionProxy is an Abstract UI Description for the first method of the selected Web Service. This description is returned to the ServiceClient via its *notifyUIReceived()*-method and again forwarded to an *AbstractUIDParser* via its *parseUID()*-method. This class takes the Abstract UI Description, parses it and generates an object oriented representation of it called *UIObject*. This object contains the same information as the XML-description, but is easier to handle and provides more flexibility.
In the next step, an interface for the invocation of the selected Web Service is created from the UIObject: The ServiceClient creates a *GenericForm* object which provides the scaffolding for the generation of an anonymous user interface with specific widgets from the information that is encapsulated in the UIObject. Next, the ServiceClient transfers the GenericForm with the generated interface to the *showAndSetDisplayable()*-method of the PerciClientMidlet. This method is another central part of the PerciClientMidlet's application logic. The goal of its generic interface-update-mechanism is to dynamically update anonymously generated interfaces upon the retrieval of parameter values through Physical Mobile Interaction. The manageTags()-method implements a generic mechanism to get interaction events through various interaction techniques. All interaction techniques that want the application to use their interaction events, have to forward them to the PerciClientMidlet through its manageTags()-method.
Similarly, all interface classes that want to be notified about new interaction events in order to eventually use them for updating their widgets with the corresponding parameter values, have to implement the *UpdateInterface*-class and it *update()*-method.
The showAndSetDisplayable()-method has the PerciClientMidlet display the forwarded interface - mostly a GenericForm - and remember it as the currently set displayable. The next time, the InteractionClient receives a parameter-tag through Physical Mobile Interaction and forwards it to the manageTags()-method, this tag is given to the update()-method of the currently set displayable, in case of Figure 6.10 the GenericForm. The GenericForm uses the information from the parameter-tag and updates its widgets accordingly.
When all parameters that are necessary for the invocation of the interface's Web Service have been gathered, the GenericForm calls the invokeHttpConnection()-method of the ServiceClient in order to send the parameters to the service. This cycle of interface updates and subsequent service invocations is repeated until the last method of a service has returned a result or until the next service is selected in which case the process starts again from the beginning. The next chapters will present the introduced objects, interactions and mechanisms on code level and in more detail.

6.3 Interacting with the Real World

The InteractionClient class represents the Physical Mobile Interaction Domain in the prototype application as it manages and encapsulates the interaction techniques Touching and Pointing. Both of them were implemented using the corresponding packages from PMIF in order to use the features of NFC and visual marker recognition technology. Since Touching and Pointing can't be used together on the same mobile phone (see chapter 6.2.1), two slightly different prototypes were created. Each of them supports either Touching or Pointing on either the Nokia 3220 or the Nokia 6630. Direct Input is included with both prototypes. This chapter shows in detail, how these 3 interaction techniques were implemented and how they provide the application with information through Physical Mobile Interaction. A particular focus is put on the implementation and use of the PMIF NFC-package, since it has especially been implemented for the prototype application and is its main interaction technique.

6.3.1 Implementing Touching with NFC

Starting the Interaction with the PMIF framework

The implementation of the Physical Mobile Interaction technique Touching uses NFC in order to receive information from selected objects. As a part of the application's interaction design, this technique does not require any explicit activation, has no interface for its invocation and is constantly available from anywhere in the application. Users can apply it implicitly by simply touching options on the posters with the NFC shell of the Nokia 3220 (Figure 6.9a, b) in order to interact with it.

Touching is automatically started together with the InteractionClient which is itself created and started by the PerciClientMidlet, the main class of the prototype application. The InteractionClient extends the Thread class in order to anticipate interaction events concurrently with the main thread without disturbing it. Its run()-method (Figure 6.11, line 01) registers a newly created *NFCInteractionController* and the corresponding NFC-InteractionType with the InteractionManager in order to prepare the connection to the NFC shell. The InteractionManager has previously been created in the constructor of the InteractionClient.

```
01    public synchronized void run()
02    {
03      manager.register(new NFCInteractionController(), InteractionTypes.NFC);
04      manager.setInteractionListener(this, InteractionTypes.NFC);
05      ...
06      startInteraction(InteractionTypes.NFC);
07    }
08    ...
09
10    public void startInteraction(String interactionType)
11    {
12      manager.startInteraction(interactionType);
13    }
```

Figure 6.11: The methods run() and startInteraction() from the InteractionClient class

The InteractionClient implements the InteractionListener-interface from the pmif.interaction-package and sets itself as a listener for the NFC-InteractionType in order to be notified about the establishment of a PhysicalObjectConnection with the NFC shell. In order to receive events and data from NFC-tags and –devices, the InteractionClient also implements the *NFCListener*-interface from the *pmif.interaction.nfc*-package. The methods that are required by these interfaces, are discussed below.

In order to initiate the interaction with the NFC shell, the InteractionClient calls its own *startInteraction()*-method (Figure 6.11, lines 06 and 10) which itself calls the *startInteraction()*-method of the InteractionManager. This method finally starts the InteractionController that has previously been registered for the same InteractionType, that is provided as a parameter and

whose interaction routine is to be started by the PMIF framework. In case of the prototype application, the NFCInteractionController from Figure 6.11, line 03 is started.
Instead of calling the InteractionManger.startInteraction()-method directly, the InteractionClient masks this method behind its own public method of the same name, in order to provide public access to it. In order to use a certain interaction technique respectively InteractionController, developers can register and start them separately. This mechanism will be convenient for the invocation of other interaction techniques, e.g. Pointing (see below). After the InteractionClient has started the connection to the NFC-shell in its run()-method, Touching is permanently available for Physical Mobile Interaction.

The Classes and Mechanisms of the pmif.interaction.nfc-Package

The NFCInteractionController and NFCListener classes are part of the pmif.interaction.nfc-package which also includes the classes *NFCConnection* and *NFCEventListener*. The classes NFCInteractionController and NFCConnection are subclasses of the generic core PMIF-classes InteractionController and PhysicalObjectConnection (see chapter 5.4.3). The NFCInteractionController manages the other objects and its mechanisms that are necessary to establish a connection to the Nokia NFC shell as well as to read from and write to that connection.
Managed by the PMIF-framework, the invocation of the InteractionManager.startInteraction()-method (Figure 6.11, line 12) calls the startInteraction()-method of the NFCInteractionController class (Figure 6.12). This method creates a new NFCEventListener and a new *ContactlessConnection* which is the actual connection to the NFC shell and which requires 3 parameters: An object of the *ContactlessListener* class, the features of the ContactlessConnection, the application is interested in, and the URL of the connection target. If this last parameter is set to *null* (Figure 6.12, line 11), the connection can be established with the NFC shell. Nevertheless the parameter can also be set to “socket://127.0.0.1:7618” (Figure 6.12, line 07), a port on the localhost computer where the Nokia Cover Emulator – another part of Nokia’s NFC/RFID package [nffs] – waits and emulates the NFC shell for testing purposes. The values for the second parameter – “ntip_container” and “nfc_device” – indicate that the application is interested in data and events from other NFC devices and NFC tags.

```
01      public void startInteraction()
02      {
03        nfcEventListener = new NFCEventListener(this, (NFCListener)listener);
04
05        //connect to cover emulator
06        conn = ContactlessConnection.openConnection(nfcEventListener, new String[]{"ntip_container","nfc_device"},
07        "socket://127.0.0.1:7618");
08
09        //connect to nokia nfc shell
10        //conn = ContactlessConnection.openConnection(nfcEventListener,
11        //new String[]{"ntip_container","nfc_device"}, null);
12      }
```

Figure 6.12: The startInteraction()-method from the NFCInteractionController class

The first parameter of the constructor requires an object of type ContactlessListener, an interface that must be implemented in order to receive *ContactlessEvents* - data and events from the selected features of the ContactlessConnection. This interface is implemented by the NFCEventListener which manages the distribution of all ContactlessEvents in the framework.
The NFCInteractionController creates an NFCEventListener object and provides the two objects that are interested in ContactlessEvents as the parameters of the constructor: itself and an object of type NFCListener. The listener-object from line 03 of Figure 6.12 is commonly provided by the InteractionController class and implements the InteractionListener-interface. This object is used to return acquired PhysicalObjectConnections to their designated listeners (see below). In the prototype application, this object is the InteractionClient which has registered itself as an InteractionListener for the NFC-InteractionType (see Figure 6.11, line 04). The listener-object can be cast into a NFCListener, since the InteractionController also implements this interface.

The NFCListener-interface provides three methods (Figure 6.13) – *notifyTagRead()*, *notifyDeviceRead()* and *notifyStatusChanged()* – that forward ContactlessEvents or data from NFC tags and NFC devices to the implementing class.

```
01    public interface NFCListener
02    {
03      //notifies applications about data read from nfc-tags
04      public void notifyTagRead(NTIPRecord[] records);
05
06      //notifies applications about data read from nfc-devices
07      public void notifyDeviceRead(NTIPRecord[] records);
08
09      //forwards ContactlessEvents to applications
10      public void notifyStatusChanged(ContactlessEvent event);
11    }
```

Figure 6.13: The NFCListener-interface and its methods

After the NFCInteractionController has created the ContactlessConnection, it has the NFCEventListener wait for a response from that connection. The functionality of listening for ContactlessEvents has been outsourced into the independent NFCEventListener class in order to separate functionalities and responsibilities from each other and to avoid their mixup. The NFCEventListener class receives ContactlessEvents through the handleEvent()-method (Figure 6.14) which is inherited from the ContactlessListener-interface. The switch-block of this method catches certain ContactlessEvents and decides what to do with them.

```
01    public void handleEvent(ContactlessEvent e)
02    {
03      listener.notifyStatusChanged(e);
04
05      switch(e.getType())
06      {
07        case ContactlessEvent.TYPE_CONNECTION_STATUS:
08        controller.notifyConnected((e.getStatus() == ContactlessEvent.STATUS_CONNECTED));
09        break;
10
11        case ContactlessEvent.TYPE_READ_NTIP:
12        {
13          if (e.getStatus() == ContactlessEvent.STATUS_OK)
14            listener.notifyTagRead((NTIPRecord[])e.getData());
15        }
16        break;
17
18        case ContactlessEvent.TYPE_RECEIVE_DATA:
19        {
20          if (e.getStatus() == ContactlessEvent.STATUS_OK)
21            listener.notifyDeviceRead((NTIPRecord[])e.getData());
22        }
23        break;
24      ...
25      }
26    }
```

Figure 6.14: The handleEvent()-method from the NFCEventListener class

First, all ContactlessEvents are forwarded to the listener-object, respectively the InteractionClient through the notifyStatusChanged()-method (Figure 6.14, line 03). The InteractionClient can use this events for its own application logic. When data has been read from either a NFC tag or a NFC device, it is also forwarded to the InteractionClient, using its inherited notifyTagRead()- and notifyDeviceRead()-methods. However, before this can happen, the NFCEventListener has to receive a ContactlessEvent of type CONNECTION_STATUS that tells about the establishment of the ContactlessConnection which was initialised by the NFCInteractionController (see Figure 6.12). The status of this event is handed back to the controller using its generic *notifyConnected()*-method (Figure 6.14, line 08 and Figure 6.15).

```
01    public void notifyConnected(boolean connected)
02    {
03      if(connected)
04      {
05        new Thread()
06        {
07          public void run()
08          {
09            NFCConnection nfcConn = new NFCConnection(conn, display);
10            listener.connectionDetected(nfcConn, InteractionTypes.NFC);
11          }
12        }.start();
13      }
14      else if(!connected)
15        listener.interactionCompleted(InteractionListener.DETECTION_ERROR);
16    }
```

Figure 6.15: The notifyConnected()-method from the NFCInteractionController class

If the ContactlessConnection has been successfully established, the notifyConnected()-method creates a new *NFCConnection* object (Figure 6.16) and forwards it to the listener-object respectively the InteractionClient through its *connectionDetected()*-method (Figure 6.15, line 10 and Figure 6.17) that has been implemented as a part of the InteractionListener-interface. This method receives the given PhysicalObjectConnection object, checks its type and casts it accordingly into a NFCConnection object for further use.

```
01    public NFCConnection(ContactlessConnection cc, Display display)
02    {
03      ...
04      nfcDevice = (NFCDevice) conn.getFeature("nfc_device");
05      ntipContainer = (NTIPContainer) conn.getFeature("ntip_container");
06    }
07
08    public void sendTitle(String string)
09    {
10      nfcDevice.send(new NTIPRecord[] {new Title(string, "en")});
11    }
12
13    public void writeTitle(String string)
14    {
15      ntipContainer.writeRecords(new NTIPRecord[] {new Title(string, "en")});
16    }
17
18    public void startReading()
19    {
20      nfcDevice.read(this);
21      ntipContainer.readRecords(this);
22    }
23    …
```

Figure 6.16: Methods from the NFCConnection class

```
01    public void connectionDetected(PhysicalObjectConnection conn, String type)
02    {
03      if(type == InteractionTypes.NFC)
04      {
05        nfcConn = (NFCConnection)conn;
06        nfcConn.startReading();
07      }
08    …
09    }
```

Figure 6.17: The NFC-routine of the InteractionClient's connectionDetected()-method

The NFCConnection class implements the PhysicalObjectConnection-interface, encapsulates the ContactlessConnection-object and represents the connection to the NFC features of the mobile phone. It also provides several methods to read from, write to and generally handle the specified features of the ContactlessConnection (see Figure 6.12, line 06 and Figure 6.16).

Using NFC for Physical Mobile Interaction

After the InteractionClient has used the PMIF framework to initiate the establishment of a connection to the NFC shell, it receives a NFCConnection object through its connectionDetected()-method (Figure 6.17). It starts reading from the NFCConnection by calling its *startReading()*-method (Figure 6.16, line 18), which again calls the corresponding reading-methods on the different NFC-features. Since the InteractionClient implements the NFCListener-interface and is registered with the NFCEventListener, it receives all ContactlessEvents and data that are generated by or read from the ContactlessConnection. In these cases, the NFCEventListener calls the NFCListener-methods, that have been implemented by the InteractionClient. In the current prototype application, ContactlessEvents are just printed out to the console for evaluation. Data that has been received from NFC tags, is provided as a NTIPRecord-array and is processed by the notifyTagRead()-method.
Data stored on NFC-tags describe the corresponding tag on the poster and its properties. Figures 6.18 and 6.19 show XML-descriptions of an action- and a parameter-tag which are separated by their different *type*-attributes. All tags share the *label*- and *desc*-elements which contain the label of the tag on the poster and an additional description. In addition, action-tags have *guidance*-elements about how to use the tag, *actiontype*-elements that distinguish between different services, e.g. the integrated Tag Detail Viewer or Web Services, as well as *actionvalue*-elements, which contain the name of the integral service or the URL of the Web Service. Parameter-tags additionally provide a *value* and its *abstractType*. The values of the abstractType-elements correspond to the Abstract Parameter Types from the Web Service descriptions (see chapter 5.2.2) which will be important for updating generated interfaces. For the prototype application, similar descriptions have been created for all tags and have been written on the corresponding NFC-tags that are attached to the posters.

```
01    <tag type="action">
02      <label>Movie Ticketing Service</label>
03      <desc>This is a service for buying movie tickets.</desc>
04      <guidance>Select options on the poster in order to buy your movie tickets.</guidance>
05      <actiontype>webservice</actiontype>
06      <actionvalue>http://perci.medien.ifi.lmu.de:8080/axis/serviceDescription/extendedCinema/</actionvalue>
07    </tag>
```

Figure 6.18: XML-description of an action-tag

```
01   <tag type="parameter">
02     <abstractType>http://perci.medien.ifi.lmu.de:8080/axis/domain/cinema/cinema.owl#MovieTitle</abstractType>
03     <value>XMen 3</value>
04     <label>XMen 3</label>
05     <desc>The X-Men make a last stand in the war between humans and mutants.</desc>
06   </tag>
```

Figure 6.19: XML-description of a parameter-tag

The Interaction receives this data through its notifyTagRead()-method (Figure 6.20) as a NTIPRecord-array and searches for a Title-object, that contains the XML-description of the tag. This description is parsed by a *TagParser*-object. The TagParser extracts all information from the tag's XML-description and creates either an *ActionTag*- or a *ParamterTag*-object from it. The ActionTag and ParameterTag classes are subclasses of the generic Tag class and contain the same information as the XML-description stored on the NFC-tags, only in object-oriented form. The Tag-object that is returned by the TagParser is forwarded to the PerciClientMidlet's manageTags()-method which uses the information from this object for its application logic.

```
01    public void notifyTagRead(NTIPRecord[] records)
02    {
03      Title t = null;
04
05      for (int i = 0; i < records.length; i++)
06      {
07        if (records[i] instanceof Title)
08        {
09          t = (Title) records[i];
10          Tag tag = parser.parseTag(t.getTitle());
11          midlet.manageTags(tag);
12        }
13      }
14    }
```

Figure 6.20: The notifyTagRead()-method from the InteractionClient class

6.3.2 Implementing Pointing with Visual Markers

The Physical Mobile Interaction technique Pointing uses the recognition of visual markers in order to select objects and interact with them. Its implementation is based on the classes of the PMIF *pmif.interaction.visual*-package that has been created by Sergej Wetzstein [RWS 05]. Due to the generic mechanisms of PMIF, the application of these components is very similar to Touching. However, Pointing requires its own interface for its service and is not automatically started together with the InteracionClient. Instead, the Pointing technique has to be called explicitly via the “Capture”-command from the menus of the Main Screen, the Generic Form and the Tag Detail Viewer in order to select action- or parameter-tags.

The initialisation of Pointing is done the same way as with Touching: The InteractionClient creates a new *VisualTagInteractionController* object in its run()-method, registers it with the InteractionManager for the Visual-InteractionType and also sets itself as a listener for this InteractionType. Opposite to Touching, setting up and starting the particular InteractionController are separated from each other. Since Touching is intended to be available for implicit interaction at all times, its InteractionController is automatically started at the end of the run()-method using the InteractionClient’s startInteraction()-method. Pointing has to be started explicitly from a menu in order to be used, since its required interface makes it less convenient for spontaneous use. Therefore it can also not be constantly available and has to be restarted for every interaction. Nevertheless, it can be easily started from everywhere in the application calling the InteractionClient’s startInteraction()-method with the Visual-InteractionType as its parameter.

The same way the NFCInteractionController manages NFC-functionalities, the VisualTagInteractionController is responsible for setting up and managing the mobile phone camera in order to use it for the recognition of visual markers. Upon starting this class, a camera screen is displayed (see Figure 6.3a) that can be used for taking pictures. After the user has pressed the “Recognize”-button, a new *VisualTagConnection* object is created from the taken picture and returned to the InteractionClient via its connectionDetected()-method (Figure 6.21). Similar to the NFCInteractionController, the InteractionClient is available through the listener-object that implements the InteractionListener-interface and has set itself as a listener for PhysicalObjectConnections of type InteractionType.VISUAL.

The VisualTagConnection class is a subclass of PhysicalObjectConnection, encapsulates the raw image data that was returned from the camera after taking a picture, provides methods to read its data and sets a recogniser for analysing the contained visual code.

The connectionDetected()-method of the InteractionClient receives the VisualTagConnection object the same way it received the NFCConnection (Figure 6.21). It checks its type and casts it accordingly into a VisualTagConnection object. Afterwards, it sets a newly created VisualCodeTagRecognizer for the VisualTagConnection. The VisualCodeTagRecognizer uses an object of type CompactRecognizer that is also included with PMIF in order to extract the visual code from the picture of the visual marker.

```
01      public void connectionDetected(PhysicalObjectConnection conn, String type)
02      {
03        …
04        if (type == InteractionTypes.VISUAL)
05        {
06          if (tagRecognizer == null)
07            tagRecognizer = new VisualCodeTagRecognizer();
08
09          visualConn = (VisualTagConnection)conn;
10          visualConn.setRecognizer(tagRecognizer);
11          String message = visualConn.readVisualTag();
12
13          if(message != null)
14          {
15            Tag tag = tagStorage.getTag(message);
16            midlet.manageTags(tag);
17          }
18        }
19        ...
        }
```

Figure 6.21: The visual marker-routine of the InteractionClient's connectionDetected()-method

Due to the limited amount of data that can be coded into a visual marker, the extracted visual code can only identify the corresponding tag on the poster, but it can't hold the data that is necessary to describe it. NFC tags can directly store this data as a XML-description. In order to provide the same information for successful Physical Mobile Interaction, the codes from visual markers have to be mapped to the appropriate information that is associated with the corresponding tags. This mapping between visual code and tag-information can be achieved by sending the visual code to a server and have the appropriate information sent back to the client application. The application prototype uses another way, that makes testing easier and less dependent on having to connect to a server. The *TagStorage* class implements a hashtable that stores references to local XML-files containing the tag information and uses the visual codes as its keys. The information about a tag is returned as a Tag object that has been created from the XML-description of the tag.
The connectionDetected()-method takes the visual code that is returned by the VisualCodeTagRecognizer, passes it to the TagStorage and receives the corresponding Tag object which is again forwarded to the PerciClientMidlet's manageTags()-method for further processing.

6.3.3 Implementing Direct Input

The Direct Input interaction technique was not implemented with PMIF and is hence not managed by the InteractionClient. The reason for this decision is the PMIF implementation of this technique in the *pmif.interaction.userinput*-package, which provides its own inflexible interface for user-mediated input which does not conform to the application's interaction design. Most of the interfaces that are generated from Web Service interface descriptions handle Direct Input themselves, since they are composed of common J2ME widgets. Therefore a specific interface for Direct Input is only required for selecting action-tags respectively services which leads to the generation of the other service interfaces. Furthermore, a Direct Input interface is also necessary in order to select movie-title tags for the integral (Movie) Tag Detail Viewer.
For these last two use cases, the prototype application uses the *DirectInputScreen* class which extends the Form class and provides different sets of common J2ME interface widgets according to its task. The DirectInputScreen for selecting different services is called from the Main Screen menu and the interface for selecting movie-titles in order to learn more about their details is shown after the Action Info Screen of the Tag Detail Viewer-service. The implementation of the DirectInputScreen itself is pretty straightforward: Its constructor uses a *type*-parameter in order to differentiate between 3 different interfaces (Figure 6.22). The first two options merely list the available services and movie-titles as exclusive radio-button choices for easier testing. For more flexible direct textual input, these radio-button-lists can easily be

replaced by the third interface which is a common TextField widget. As a result of this choice, service parameter values and service-URL that are eventually provided by the poster, have to be typed by hand.
Apart from the code to assemble the interface, the DirectInputScreen also checks the existence and the correctness of the provided input – especially in case of the TextField input widget. Similar to Pointing, the information acquired through Direct Input only identifies services and parameters, but does not describe it in a way, that is sufficient for the application's management of Physical Mobile Interaction. Direct Input can rely on the same options to solve this problem as Pointing: retrieving missing information from a server or from within the application itself. The current implementation uses the second solution and gets tag descriptions from a TagStorage object that stores them not only with visual codes but also with Direct Input identifiers. The acquired tag is again forwarded to the PericClientMidlet's manageTags()-method.

```
01      …
02      if(type == ACTIONS)
03      {
04        choiceGroup = new ChoiceGroup("Select an Action", Choice.EXCLUSIVE);
05        choiceGroup.append("Movie Ticket", null);
06        ...
07        append(choiceGroup);
08      }
09      else if(type == MOVIES)
10      {
11        choiceGroup = new ChoiceGroup("Select a Movie", Choice.EXCLUSIVE);
12        choiceGroup.append("X-Men 3", null);
13        ...
14        append(choiceGroup);
15      }
16      else if(type == UNSPECIFIED)
17      {
18        append(new TextField("Free Input", "", 150, TextField.ANY));
19      }
20      …
```

Figure 6.22: Different interfaces for the DirectInputScreen

6.4 Application Logic

The PerciClientMidlet is the central class of the prototype implementation and contains important parts of its application logic. It creates and intermediates between InteractionClient and ServiceClient objects that represent and encapsulate features from the Physical Mobile Interaction Domain and the Web Service Domain. It represents their intersection that realizes the use of Physical Mobile Interaction for the invocation of Web Services through dynamically generated interface. And it provides and manages common objects and features that implement this intersection in order to guarantee the overall operation and correctness of the application. The PerciClientMidlet works in the background of the client application and does not provide its own interface. Instead, it uses an object of type *MainScreen* as the main interface of the application to which most of the other interfaces can return to.
As mentioned in chapter 6.2.2, the PerciClientMidlet implements a generic interface-update-mechanism that forwards service- and parameter-values to anonymously generated service interface in order to update them dynamically. At the heart of this mechanism is the manageTags()-method (Figure 6.23). After the PerciClientMidlet has been created, this method waits for Tag objects that are selected through the supported Physical Mobile Interaction techniques (see chapter 6.3). The manageTags()-method acts as an interface to these techniques since they have to use it in order to have their acquired information processed by the application. This method drives the application's logic by deciding what to do with the received Tag objects.

In order to guarantee the correctness of the update-mechanism and the association between selected services and parameters, the PerciClientMidlet uses the *action*-object (Figure 6.23, line 01) in order to store the ActionTag object that represents the currently selected action/service. Another object called *ptSession* of type *ParameterTagSession* (Figure 6.23, line 02) stores all parameter-tags that are selected during the time – the "session" – the current service is used. The purpose of this object is to remember all parameter-tags for a service in order to update all widgets of its interface. Otherwise the application could only update single widgets, while forgetting the previously selected parameter-values for the other ones (see below). The ParameterTagSession object has to be cleared after an action/service has finished before it can be used with the next one.

```
01      action = null;
02      ptSession = new ParameterTagSession();
03
04      public void manageTags(Tag tag)
05      {
06        if(tag instanceof ActionTag)
07        {
08          action = (ActionTag)tag;
09          ptSession.clearTable();
10          showFeedbackAlert("", "New Action available", new ActionInfoScreen(this, action, serviceClient));
11        }
12        else if(tag instanceof ParameterTag)
13        {
14          if(action == null)
15            showAlert("", "Please select an ActionTag first!", currentDisplayable);
16          else
17          {
18            ptSession.storeParameterTag((ParameterTag)tag);
19
20            if(currentDisplayable instanceof UpdateInterface)
21              ((UpdateInterface)currentDisplayable).update((ParameterTag)tag, ptSession);
22            else
23              showFeedbackAlert("", "New " + ((ParameterTag)tag).getLabel() + " selected", currentDisplayable);
24          }
25        }
26      }
```

Figure 6.23: The manageTags()-method from the PerciClientMidlet class

When the manageTags()-method is called by a Physical Mobile Interaction technique, it receives an object of the generic type Tag that contains and represents the description of a selected option. The method differentiates between the two different types of tags and handles them accordingly. If the received Tag object is an instance of the ParameterTag class, the manageTags()-method checks whether the action-object has already been initialised since parameter-tags can't be used without an action/service. If no ActionTag has been provided yet, an alert is displayed that makes users aware of its absence (Figure 6.23, line 15).
The PerciClientMidlet provides two types of alerts: a normal alert by calling the showAlert()-method (Figure 6.24) which creates, configures and displays a standard alert that can be cancelled through user input and a special feedback-alert by calling the showFeedbackAlert()-method (Figure 6.24) that uses the standard alert, but increases its signalling effect through additional vibration and flashing backlights.
If the action-objects has already been set, the Tag object is parsed into a ParameterTag object and stored in the ParameterTagSession for the current action/service (Figure 6.23, line 18). Next, the manageTags()-method checks whether the current *Displayable*-object implements the *UpdateInterface*-interface, which is part of the interface-update-mechanism and the overall application logic (Figure 6.23, line 20). For this purpose, the PerciClientMidlet manages an object of type Displayable called *currentDisplayable* and provides a method called *showAndSetDisplayable*() (Figure 6.24) in order to set it. This method can be called in order to display any object of type Displayable and have it registered as the currently set Displayable. In order to update (generated) interfaces with parameter-values acquired through Physical Mobile Interaction in a generic, flexible and independent way, classes have to implement the

UpdateInterface and its update()-method and set themselves as the currentDisplayable-object through the showAndSetDisplayable()-method.
The next time the manageTags()-method receives a parameter-tag, it checks whether the currentDisplayable-object implements the UpdateInterface class and forwards the parameter-tag to its update()-method which again uses the information from the tag for updating its interface. This update-mechanism has several advantages: It is generic and independent from particular interface-implementations, it requires interested interface-classes to implement only a small amount of steps and it makes the distribution of parameter-tags very easy and effective.
If the currentDisplayable-object does not implement the UpdateInterface, the manageTags()-method simply triggers a feedback-alert that confirms the selection of a parameter-tag and returns to the currentDisplayable afterwards (Figure 6.23, line 23).

```
01    public void showAndSetDisplayable(Displayable d)
02    {
03       currentDisplayable = d;
04       display.setCurrent(d);
05    }
06
07    public void showAlert(String title, String text, Displayable next)
08    {
09       Alert alert = new Alert(title, text, null, AlertType.INFO);
10       alert.setTimeout(3000);
11       alert.addCommand(alertOkCMD);
12       this.getDisplay().flashBacklight(3000);
13       display.setCurrent(alert, next);
14    }
15
16    public void showFeedbackAlert(String title, String text, Displayable next)
17    {
18       showAlert(title, text, next);
19       devManager.playSequence(TOUCH_SEQUENCE);
20    }
```

Figure 6.24: The methods showAndSetDisplayable(), showAlert() and showFeedbackAlert() from the PerciClientMidlet class

In case the received Tag object is of type ActionTag, it is accordingly cast into an ActionTag object and set as the current action (Figure 6.23, line 08) so that subsequent processes can use it. Afterwards, the ptSession-object is cleared in order to provide an empty ParameterTagSession for collecting parameter-tags for the current action. Finally, a feedback alert is displayed (Figure 6.23, line 10) in order to announce the selection of an new action/service. After the feedback alert, a newly created object of type ActionInfoScreen is created which displays further information about the selected service from the description stored in the ActionTag object. From the ActionInfoScreen, the user can continue to the actual interface of the selected action/service. For this reason, the command-handler of the ActionInfoScreen implements a routine that decides how to proceed with different services (Figure 6.25). This routine uses the information from the PerciClientMidlet's action-object and either calls an integral interface for a local service or initiates the generation of Web Service interface.
The current prototype application only includes the local Movie Tag Detail Viewer. In case this service has been selected, a new *TagDetailScreen* object is created and displayed using the showAndSetDisplayable()-method from the PerciClientMidlet (Figure 6.25, line 07). This class implements the UpdateInterface and its update()-method and provides an interface that shows details of selected movie-tags on the posters (Figure 6.5). Since it is set as the currentDisplayable object by the showAndSetDisplayable()-method, its update()-method is called by the manageTags()-method upon the next retrieval of a parameter-tag. If this tag contains information about a movie, the TagDetailScreen updates its interface and shows them.
All other services that are not integrated with the application, are Web Services whose interfaces have to be rendered from separate descriptions. In that case, the ActionInfoScreen-routine creates and shows a *WaitingScreen* object and starts the retrieval of the Web Service interface description by calling the *invokeHttpConnection()*-method of the ServiceClient.

```
01      ...
02      if(cmd == nextCMD)
03      {
04        if((action.getActionValue()).equals("viewer"))
05        {
06          TagDetailScreen vForm = new TagDetailScreen(midlet, action.getLabel());
07          midlet.showAndSetDisplayable(vForm);
08        }
09        else if((action.getActionType()).equals("webservice"))
10        {
11          midlet.showDisplayable(new WaitingScreen(midlet, null, "Please wait", ""));
12          serviceClient.invokeHttpConnection(action.getActionValue(), null, "GET");
13        }
14      }
15      ...
```

Figure 6.25: Code fragment from the ActioInfoScreen-class

6.5 Interface Generation, Rendering and Update

Next to the InteractionClient and the PerciClientMidlet classes, the ServiceClient is the third important class of the prototype implementation core. It manages the communication between the mobile client application and the Interaction Proxy and coordinates the interface generation process and its components. Figure 6.26 outlines the different steps involved:

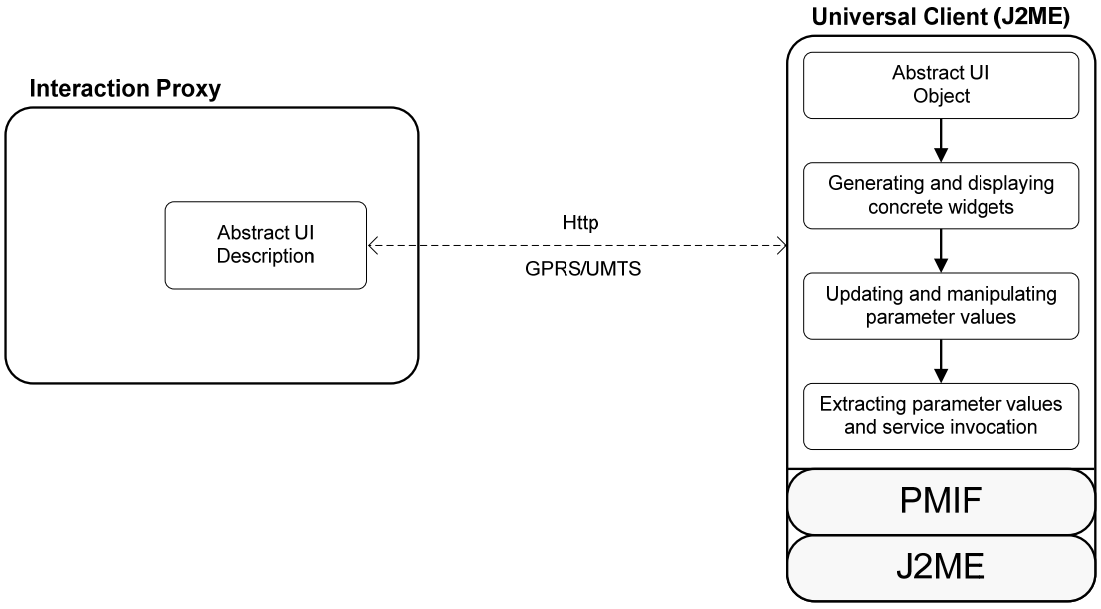


Figure 6.26: The different steps of interface generation

The Universal Client implementation uses the Service Client (not depicted in Figure 6.26) in order to communicate with the Interaction Proxy via HTTP over GPRS or UMTS, invoke services and request service interface descriptions, which usually goes hand in hand, except for the first request for the initial user interface. Focusing on the interface generation process, the Interaction Proxy returns the Abstract UI Description of a service. This description is received by the Service Client which generates an Abstract UI Object from it and uses it for the generation and rendering of a generic interface with concrete widgets. Afterwards these widgets are manipulated through Physical Mobile Interaction and updated with the thereby acquired

parameter-values. After the interaction has been finished, the acquired parameter-values are extracted from their widgets and serve as request-parameters for the invocation of the next service and the retrieval of its interface.
In order to manage all these different yet correlated steps of the interface generation cycle - service invocation, interface description retrieval, interface generation, rendering, manipulation and update – the ServiceClient class coordinates several components and functionalities. It provides the invokeHttpConnection()-method (Figure 6.27) that is called from different classes of the prototype implementation in order to establish a connection to the Interaction Proxy. The method basically creates and runs a separate *HttpServiceConnection*-object (Figure 6.27, line 04, 06) that encapsulates the communication with the Interaction Proxy and the thus masked, indirect invocation of Web Services. The HttpServiceConnection-class runs in a separate thread, establishes and manages different http-connections with the Interaction Proxy. For that purpose it uses delivered parameters that specify the URL of the Interaction Proxy, URLs of specific Web Services from action-tags, session-ids, request-method and other request parameters for the invocation of specific services. Finally it receives the response for a request from the Interaction Proxy, creates a string from it and returns it to the ServiceClient by calling its *notifyUIReceived()*-method (Figure 6.27, line 10).

```
01    public void invokeHttpConnection(String serviceURL, String[][] parameters, String method)
02    {
03      ...
04      httpConn = new HttpServiceConnection(this, serviceURL, parameters, method, midlet);
05      ...
06      httpConn.run();
07      ...
08    }
09
10    public void notifyUIReceived(String uiDescription)
11    {
12      …
13      uiObject = parser.parseUIDescription(uiDescription);
14      genericForm = new GenericForm(this, uiObject, midlet);
15      midlet.showAndSetDisplayable(genericForm);
16      …
17    }
```

Figure 6.27: The invokeHttpConnection()-method from the ServiceClient class

The response from the Interaction Proxy that is forwarded to the notifyUIReceived()-method, contains the Abstract UI Description of a Web Service interface. The ServiceClient uses an *AbstractUIDParser* in order to parse it into an UIObject. Similar to a TagParser and XML-tag-descriptions (see chapter 6.3.1), the AbstractUIDParser transforms an Abstract UI Description into an UIObject that contains the same information as the XML-description, only as an object-oriented representation. An UIObject basically contains and manages several *Widget*-objects. Each of them refers to a widget-element from the Abstract UI Description and contains exactly the same information about labels, abstract types or value sets (see Figure 5.5). The Abstract UI Description and its widgets are parsed and encapsulated in these objects for a more flexible interface generation and easier access to required information than provided by frequent XML-parsing.
After the UIObject has been created from the Abstract UI Description, a newly created GenericForm-object uses its information for the generation of the final user interface (Figure 6.27, line 14). The GenericForm class provides a generic frame, a container that comprises and manages different widgets and their generation. Its constructor iterates over all Widgets-objects that are stored in a hashtable of the UIObject (Figure 6.28, line 02) and maps them to concrete J2ME-widgets according to the Abstract Widget Type of the Widget-objects. The value of the Abstract Widget Type gives a hint for the usage of the widget and thus the mapping to concrete J2ME-widgets. Widget-objects with a "*directInputParameterType*" are mapped to J2ME-*TextField*-objects for arbitrary input (Figure 6.28, line 06), *ChoiceGroups* are created for "*singleSelectInputParameterType*"-widgets for single selections from several options (Figure

6.28, line 12) and all "*plainOutputParameterType*"-widgets are mapped to *StringItem*-objects that only display text messages (Figure 6.28, line 06). All created widgets are appended to the GenericForm-object that extends the Form-class and can thus be displayed on the mobile client screen.

```
01      …
02      for(Enumeration e = widgetTable.elements(); e.hasMoreElements();)
03      {
04        Widget widget = (Widget)e.nextElement();
05
06        if(widget.getParameterType().equals("directInputParameterType"))
07        {
08          TextField field = new TextField(widget.getLabel(), "", 50, TextField.ANY);
09          itemRefTable.put(widget.getAbstractType(), field);
10          append(field);
11        }
12        else if(widget.getParameterType().equals("singleSelectInputParameterType"))
13        {
14          ChoiceGroup choiceGroup = new ChoiceGroup(widget.getLabel(), Choice.EXCLUSIVE);
15          ParameterValueOption[] array = widget.getParameterValueSet();
16
17        for(int i=0; i<array.length; i++)
18        {
19          choiceGroup.append(array[i].getLabel(), null);
20        }
21
22          itemRefTable.put(widget.getAbstractType(), choiceGroup);
23          append(choiceGroup);
24        }
25        else if(widget.getParameterType().equals("plainOutputParameterType"))
26        {
27          StringItem stringItem = new StringItem(widget.getLabel(), widget.getDescription());
28          itemRefTable.put(widget.getAbstractType(), stringItem);
29          insert(0, stringItem);
30        }
31      }
32      ..
```

Figure 6.28: The constructor of the GenericForm-class

The presented interface generation process is very generic as it dynamically creates and assembles widgets from rather abstract descriptions. Hence the created widgets are anonymous and hard to reference which is nevertheless important to manipulate and update them later on. Therefore all anonymously generated interface widgets are stored in a hashtable called *itemRefTable* for future references using the Abstract Parameter Type of the widget description, respectively the *abstractType*-variable of the Widget-object as the key (see e.g. Figure 6.28, line 28).

After the GenericForm and its widgets have been created and assembled, the notifyUIReceived()-method of the ServiceClient forwards this interface-class to the PerciClientMidlet using its showAndSetDisplayable()-method (Figure 6.27, line 15). The GenericForm and its widgets are displayed on the mobile device screen and set as the current Displayable of the application. Since the GenericForm implements the UpdateInterface-interface, it is now automatically registered with the update-mechanism of the application logic (see chapter 6.4). As soon as the manageTags()-method of the PerciClientMidlet receives a new ParameterTag through Physical Mobile Interaction, the ParameterTagSession-object that stores all ParameterTags for an action respectively for a service - including the new ParameterTag - is forwarded to the GenericForm through its implementation of the update()-method (see Figure 6.23, lines 18, 21).

This implementation of the update()-method in the GenericForm (Figure 6.29) uses the ParameterTags from the current ParameterTagSession in order to update its widgets with the latest parameter-values as a reaction to the Physical Mobile Interaction which acquired these values. For this purpose, it matches all ParameterTags of the ParameterTagSession against all widgets of the interface that have been registered in the itemRefTable-object. This matching process uses the Abstract Parameter Types of both of them as the matching criteria in order to

allocate information from ParameterTags to the correct widget of the corresponding Abstract Parameter Type. The update()-method iterates over all ParameterTags from the forwarded ParameterTagSession, extracts its Abstract Parameter Type and uses it to find a matching Widget-object with the same Abstract Parameter Type from the itemRefTable-object (Figure 6.29, lines 04 - 11). That way, information from ParameterTags only updates widgets for which the information has been selected through Physical Mobile Interaction.

```
01      public void update(ParameterTag latestTag, ParameterTagSession session)
02      {
03        ...
04        for(Enumeration e = ptSession.elements(); e.hasMoreElements();)
05        {
06          ParameterTag pTag = null;
07
08          if(e.nextElement() instanceof ParameterTag)
09            pTag = (ParameterTag)e.nextElement();
10
11          Object object = itemRefTable.get(pTag.getAbstractType());
12
13          //handle TextField-widgets
14          if(object != null && object instanceof TextField)
15          {
16            TextField field = (TextField)object;
17            field.setString(pTag.getLabel());
18            String abstractType = latestTag.getAbstractType();
19            midlet.showFeedbackAlert("", "New " + abstractType + " selected", this);
20          }
21          //handle ChoiceGroup-widgets
22          else if(object != null && object instanceof ChoiceGroup)
23          {
24            ...
25          }
26        }
27        ...
28      }
```

Figure 6.29: The update()-method of the GenericForm-class

After a widget that matches a certain Abstract Parameter Type has been retrieved and cast from the itemRefTable, its classification of a concrete J2ME-widget is verified and its value is updated with the information from the ParameterTag. A feedback alert finally announces the selection of a the latest selected parameter and the update of the corresponding interface widget which is subsequently displayed. Figure 6.29 depicts these steps for TextField-object on lines 14 to 19.

This process of updating the GenericForm and its widgets through Physical Mobile Interaction is repeated until all parameters for the invocation of the corresponding service have been collected. Users can initiate this invocation by pressing a “Send”-button from the menu of the GenericForm. Its command handler reacts to this event and reads the current values from all widgets in the itemRefTable-object. If all widgets can provide legal values, these values together with their corresponding Abstract Parameter Types are forwarded to the invokeHttpConnection()-method of the ServicClient and used as parameters for the invocation of the next service-method. With this invocation of the invokeHttpConnection()-method, the cycle of service invocation, retrieval of the Abstract UI Description for the next interface, its generation, rendering and manipulation starts again, until all steps of a service and their interfaces have been finished.

Chapter 7

Conclusion and Future Work

This diploma thesis presented, discussed and evaluated a new generic approach to more complex Physical Mobile Interaction through the use of Semantic Web Service descriptions for the automatic generation of adaptable user interfaces that support and facilitate the mobile interaction with physical objects and their associated digital information and services. In collaboration with Sven Siorpaes and his diploma thesis [Sior 06], one goal of this thesis was to design a framework that exploits the flexibility and expressiveness of Semantic Web Service descriptions for the dynamic generation of user interfaces that can be adapted to client-, device- and other context information. This interface generation process provides the foundation for more sophisticated Physical Mobile Interaction, which is not only beneficial for the more intuitive invocation of the framework's own Web Services, but also advances and overlaps with other technologies like the Internet of Things.

While most of the proposed framework and its services were developed and implemented by Sven Siorpaes (see his diploma thesis [Sior 06] for details), this diploma thesis built upon the framework in order to realise more complex yet intuitive Physical Mobile Interaction. It focused on the design, implementation and evaluation of a prototype client application that acts as a front end of this framework and thus intermediates between its services and physical objects in the real world through the implementation of Physical Mobile Interaction techniques.

The overall acceptance of this approach has been confirmed by a preliminary user study that was conducted in the context of 2 use case scenarios for mobile ticketing, for which 2 posters were designed as part of this diploma thesis. These posters provided different options for ordering movie and public transportation tickets, which can be selected through Touching, based on Near Field Communication technology, Pointing, based on the recognition of visual markers, and Direct Input that is implicitly supported by common interface widgets. After the user study, the mobile client application was implemented in order to support these interaction techniques and thus allow the usage of the posters and their associated services.

Both the implementation of the system framework and the mobile client application showed that the proposed approach to Physical Mobile Interaction through the generation of interfaces from Semantic Web Service descriptions is technically feasible and reasonable. The current system provides generic means to dynamically generate reusable HTML- and J2ME-interfaces that are automatically adapted to user- and device-context. Although the interaction techniques Touching and Pointing could not be used together due to the lack of mobile device support, the

mobile client application is able to support different, exchangeable Physical Mobile Interaction techniques.
After the implementation phase, a final user study was conducted in order to evaluate the acceptance and effectiveness of the whole system and especially of the client application and the 3 interaction techniques it supported. As for the results of this user study, the approach to and usage of Physical Mobile Interaction was generally very well received, despite an initial inhibition level due to the unfamiliar concept of different action- and parameter-tags and the new type of interaction in general. Concerning the different Physical Mobile Interaction techniques, Touching was the clear favourite in respect to usability, innovation and reliability. Pointing on the other hand was perceived as the exact opposite of Touching: more complicated to use, less reliable and less convenient. Direct Input benefited from its easy handling and great reliability but was not considered to be innovative or interesting.
Despite the generally great acceptance of Physical Mobile Interaction and the 3 presented techniques it also became evident from the user study that the potential of Physical Mobile Interaction is not yet fully exploited and needs more support. This showed especially as subjects of the user study did not use the poster and the application as intuitively and naturally as Physical Mobile Interaction would allow them to do (see [Sior 06] for more details).

While the current implementation of the framework and its mobile client application prove the feasibility of the provided concept, they are only first steps in this direction and many issues of future work remain:

- More support for different client technologies other than HTML and J2ME.
- Exploitation of Semantic Web Service reasoning and composition which was mentioned as a possible part of the framework but has not been implemented in order to keep the system simple at first.
- Using more Semantic Web technology in order to organize information and workflows within the framework.
- Exploring the idea of a distributed framework architecture that is inspired by SUPPLE.
- Making Physical Mobile Interaction more complex and sophisticated. Tags could be used across different posters or serve as input for search engines that provide different services that can be invoked with the poster’s tags.
- Development of authoring tools that build upon the framework and facilitate the different steps that are needed for setting up services and connecting them to physical objects.

References

[AbPh 99] Marc Abrams, Constantinos Phanouriou. UIML: An XML Language for Building Device-Independent User Interfaces. XML '99, Philadelphia, 1999.

[axis] The Apache Software Foundation. Axis 2.0 – Apache Axis 2 – Next Generation Web Services. Last published on 05.05.2006. http://ws.apache.org/axis2/.

[BLHL 01] Tim Berners-Lee, James Hendler, Ora Lassila. The Semantic Web. 05.2001. http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2.

[bpel] IBM. Business Process Execution Language for Web Services version 1.1. Last updated on 01.02.2005. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[capr] CAMELEON Project Homepage. Last accessed on 03.07.2006. http://giove.cnuce.cnr.it/cameleon.html.

[CCMW 01] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana. Web Service Description Lanaguage (WSDL) Homepage. 2001. http://www.w3.org/TR/wsdl.

[BHK 04] Mark Butler, Johan Hjelm, Kazuhiro Kitagawa. Composite Capabilities/Preference Profile Public Home Page. 2004. http://www.w3.org/Mobile/CCPP/.

[chci] Center for HCI @ VT. Last modified on 13.06.2006. http://www.hci.vt.edu/.

[ChMo 06] Deepika Chauhan, Bradley Morrison. OpenSource.Nokia.com - S60 OSS Browser. Last accessed on 03.07.2006. http://opensource.nokia.com/projects/S60browser/.

[cldc] Sun Developer Network. Connected Limited Device Configuration (CLDC). Last accessed on 03.07.2006. http://java.sun.com/products/cldc/.

[coc] The Apache Cocoon Project. Last published on 22.03.2006. http://cocoon.apache.org/.

[Col 04] Mark Colan. Service-Oriented Architecture expands the vision of the Web Services. 21.04.2006. http://www-128.ibm.com/developerworks/webservices/library/ws-soaintro.html.

[DeAb 99] Anind Dey, Gregory Abowd. Towards a better understanding of context and context-awareness. GVU technical report GIT-GVU-99-22, College Computing, GA Institute of Technology, 1999.

[Frou 06] Max Froumentin. Device Independence Activity Homepage. Last revised on 16.05.2006. http://www.w3.org/2001/di/.

[dmi] Department of Media Informatics, LMU. Last accessed on 03.07.2006. http://www.medien.informatik.uni-muenchen.de.

[ecl] Eclipse.org home. Last accessed on 03.07.2006. http://www.eclipse.org/.

[epc] EPCglobal Inc. Home Page. Last accessed on 03.07.2006. http://www.epcglobalinc.org/.

[feli] NTT DoCoMo: Services. Last accessed on 03.07.2006. http://www.nttdocomo.com/corebiz/icw/index.html.

[GaWe 04] Krzysztof Gajos, Daniel S. Weld. Automatically Generating User Interfaces For Ubiquitous Applications. In: Workshop on Ubiquitous Display Environments, Nottingham, UK, 2004.

[gre] Mozilla Developer Center. GRE – MDC. Last modified on 08.03.2006. http://developer.mozilla.org/en/docs/GRE.

[Haas 06] Hugo Haas. Web Services Activity Statement. 21.06.2006. http://www.w3.org/2002/ws/Activity.

[harm] Harmonia, Inc. Last accessed on 03.07.2006. http://www.harmonia.com/.

[heis] Heise Online. Quelle-Werbeplakate mit Infrarot-Schnittstelle. 10.12.2005. http://www.heise.de/newsticker/meldung/67212.

[Herm 06] Ivan Herman. W3C Semantic Web. 29.06.2006. http://www.w3.org/2001/sw/.

[HSB 06] Ivan Herman, Ralph Swick, Dan Brickley. Resource Description Framework (RDF). 29.06.2006. http://www.w3.org/RDF/.

[idbl] Cathexis Innovations Inc. IDBlue. Last accessed on 03.07.2006. http://www.cathexis.com/products/idblue.aspx.

[jase] Sun Developer Network. Java Servlet Technology. Last accessed on 03.07.2006. http://java.sun.com/products/servlet/.

[j2me] Sun Developer Network. Java Platform, Micro Edition (Java ME). Last accessed on 03.07.2006. http://java.sun.com/javame/.

[jwsa] Sun Developer Network. J2ME Web Services APIs (WSA). Last accessed on 03.07.2006. http://java.sun.com/products/wsa/.

[KhLa 05] Deepali Khushraj, Ora Lassila: Ontological Approach to Generating Personalized User Interfaces for Web Services. International Semantic Web Conference 2005, 2005.

[KhTe 05] Oleksiy. Khriyenko, Vagan Terziyan. A Framework for Context-Sensitive Metadata Description, In: International Journal of Metadata, Semantics and Ontologies, 2005.

[kxml] KXML 2. Last accessed on 03.07.2006. http://kxml.sourceforge.net/kxml2/.

[LeHe 06] Philippe Le Hegaret. Web Services Activity. 29.06.2006. http://www.w3.org/2002/ws/

[Lei 06] Karin Leichtenstern. Mobile Interaction in Smart Environments. Diplomarbeit, Ludwig-Maximilians-Universität München, 2006.

[mab] MAB – Mozilla Amazon Browser. Last accessed on 03.07.2006. http://www.faser.net/mab/.

[MaLa 06] Michael Mahan, Yves Lafon. XML Protocol Workding Group. Last revised on 03.02.2006. http://www.w3.org/2000/xp/Group/.

[MBH+ 06] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, Katia Sycara. OWL-S: Semantic Markup for Web Services. 19.06.2006. http://www.daml.org/services/owl-s/1.1/overview/.

[McHa 04] Deborah L. McGuinness, Frank van Harmelen. OWL Web Ontology Language. 10.02.2004. http://www.w3.org/TR/owl-features/.

[Mel 03] Steve Meloan. Toward a Global “Internet of Things”. 11.11.2003. 2003. http://java.sun.com/developer/technicalArticles/Ecommerce/rfid/.

[mfli] Adobe. Macromedia Flash Lite. Last accessed on 03.07.2006. http://www.adobe.com/products/flashlite/.

[midp] Sun Developer Network. Mobile Information Device Profile (MIDP). Last accessed on 03.07.2006. http://java.sun.com/products/midp/

[mind] Mindswap Homepage. Last accessed on 03.07.2006. http://www.mindswap.org/.

[MPM+ 05] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, Katia Sycara. Bringing Semantics to Web Services : The OWL-S Approach. 2005. www.cs.cmu.edu/~softagents/papers/OWL-S-SWSWPC2004-final.pdf.

[moto] Mozilla 1.0 Guide: The Mozilla Toolkit. Last accessed on 03.07.2006. http://www.mozilla.org/start/1.0/guide/toolkit.html.

[moz] Mozilla.org. Last accessed on 03.07.2006. http://www.mozilla.org/.

[Mye 05] Brad Myers. Pebbles Homepage. 2005. http://www.pebbles.hcii.cmu.edu/.

[Nich 04] Jeffrey Nichols. Automatically Generating User Interfaces for Appliances. In UIST 2004 Conference Companion, Santa Fe, NM, 2004.

[nfc] NFC Forum. Last accessed on 03.07.2006. http://www.nfc-forum.org/.

[nffs] Nokia. Field Force Solutions. Last accessed on 03.07.2006. http://www.europe.nokia.com/A4149156.

[nlis] Nokia Local Interaction Server. Last accessed on 03.07.2006. http://www.europe.nokia.com/nokia/0,,76300,00.html.

[nnfc] Nokia - Nokia NFC. Last accessed on 03.07.2006. http://www.nokia.com/nfc.

[npro] Forum Nokia. Nokia Prototype SDK for Java™ Platform, Micro Edition. Last accessed on 03.07.2006. http://www.forum.nokia.com/main/0,6566,034-761,00.html.

[ntt] NTT DoCoMo Euro Labs. Last accessed on 03.07.2006. http://www.docomoeurolabs.de/.

[oma] OMA home. Last accessed on 03.07.2006. http://www.openmobilealliance.org/.

[opmi] Opera Mini 2.0. Last accessed on 03.07.2006. http://www.opera.com/products/mobile/operamini/.

[opmo] Opera Mobile. Last accessed on 03.07.2006. http://www.opera.com/products/mobile/.

[oppla] Opera Platform. Last accessed on 03.07.2006. http://www.opera.com/products/mobile/platform/.

[perc] Perci (PERvasive ServiCe Interaction). Last accessed on 03.07.2006. http://www.hcilab.org/projects/perci/.

[phil a] Philips. City of Caen, France, to demonstrate simplicity of Near Field Communication (NFC) technology. 18.10.2005. http://www.semiconductors.philips.com/news/content/file_1193.html.

[phil b] Philips. Philips, Nokia und deutscher Rhein-Main Verkehrsverbund testen NFC Handy-Ticketing. 29.04.2005. http://www.philips.at/about/news/press/halbleiter/article-15004.html.

[pml] PML The Physical Markup Language. 2002. http://web.mit.edu/mecheng/pml/index.htm;

[ps60] Forum Nokia. Python for S60. Last accessed on 03.07.2006. http://www.forum.nokia.com/python.

[RoGf 04] Michael Rohs, Beat Gfeller. Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects. In: Alois Ferscha, Horst Hoertner, Gabriele Kotsis (Eds.): Advances in Pervasive Computing, Austrian Computer Society (OCG), Vienna, Austria, 2004.

[RSH 04] Enrico Rukzio, Albrecht Schmidt, Heinrich Hussmann. Physical Posters as Gateways to Context-aware Services for Mobile Devices. Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004), English Lake District, UK, 2004.

[RWS 05] Enrico Rukzio, Sergey Wetzstein, Albrecht Schmidt. A Framework for Mobile Interactions with the Physical World. Wireless Personal Multimedia Communication (WPMC'05). Aalborg, Denmark, 2005.

[ScBG 98] Albrecht Schmidt, Michael Beigl, Hans-Werner Gellersen. There is More to Context than Location: Environment Sensing Technologies for Adaptive Mobile User Interfaces. Workshop on Interactive Applications of Mobile Computing IMC'98, 1998.

[ScTh 94] Bill Schilit, Marvin Theimer. Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8, No. 5, 1994.

[sema] Semapedia.org. Last accessed on 03.07.2006. http://www.semapedia.org/.

[Sior 06] Sven Siorpaes. A Physical Mobile Interactions Framework based on Semantic Descriptions. Diplomarbeit, Ludwig-Maximilians-Universität München, 2006.

[SRP+ 05] Albrecht Schmidt, Enrico Rukzio, Massimo Paolucci, Matthias Wagner, R. Hirschfeld. Supporting Service Interaction in the real World. Research Agreement between DoCoMo Euro-Labs and LMU. October 1, 2005.

[Stöt 01] Harald Stöttner. A Platform-Independent User Interface Description Language. Report, Johannes Kepler Universität Linz, Institut für Praktische Informatik (Systemsoftware), 2001.

[supp] University of Washington. SUPPLE: Automatic Genration of Personalizable User Interfaces. 25.09.2005. http://www.cs.washington.edu/ai/supple/.

[swtk] Sun Developer Network. Sun Java Wireless Toolkit for CLDC. Last accessed on 03.07.2006. http://java.sun.com/products/sjwtoolkit/index.html.

[tom] The Apache Software Foundation. Apache Tomcat. Last accessed on 03.07.2006. http://tomcat.apache.org/.

[uddi] Oasis. UDDI.org. Last accessed on 03.07.2006. http://www.uddi.org/.

[uiml] UIML Homepage. Last accessed on 03.07.2006. http://www.uiml.org/intro/index.htm.

[upnp] UPnP Forum. Last accessed on 03.07.2006. http://www.upnp.org.

[vico] Visual Codes. Last modified on 15.03.2006. http://www.visualcodes.org/.

[VLMB+ 04] Jean Vanderdonckt, Quentin Limbourg, Benjamin Michotte, Laurent Bouillon, Daniela Trevisan, Murielle Florins. UsiXML: A User Interface Description Language for Specifying Multimodal User Interfaces. In: Proceedings of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, 2004.

[Want 06] Roy Want. An Introduction to RFID Technology. IEEE Pervasive Computing, vol. 5, pp. 25-33, 2006.

[wiki] Wikipedia. Last accessed on 03.07.2006. http://wikipedia.org/.

[wiki a] Wikipedia. XUL. Last accessed on 03.07.2006. http://en.wikipedia.org/wiki/XUL.

[xul] Mozilla.org. XML User Interface Language (XUL). 03.07.2006. http://www.mozilla.org/projects/xul/.

[xulp] XULPlanet.com. Last accessed on 03.07.2006. http://xulplanet.com/.

[xulr] Mozilla Developer Center. XULRunner. Last modified on 15.06.2006. http://developer.mozilla.org/en/docs/XULRunner.

Annex

Supporting Service Interaction in the Real World

Gregor Broll[1], Sven Siorpaes[1], Enrico Rukzio[1], Massimo Paolucci[2], John Hamard[2], Matthias Wagner[2], Albrecht Schmidt[1]

[1] Media Informatics Group, University of Munich, Germany
{gregor, sven, enrico, albrecht}@hcilab.org
[2] DoCoMo Euro-Labs, Germany
{paolucci, hamard, wagner}@docomolab-euro.com

Abstract. Driven by improved mobile information access and capture, physical mobile interaction is increasingly gaining importance. It uses and exploits the familiar interaction with real world objects in order to provide intuitive access to associated digital information and services. This development is accommodated by the dissemination of the "Internet of Things" in which everyday objects are uniquely identified through wireless markers and have individual network references. Current implementations of physical mobile interaction with services are mostly proprietary and rather simple. Yet we think that its combination with Semantic Web services could greatly improve mobile interaction with digital information and services through associated real world objects. We introduce a conceptual architecture for exploiting semantic descriptions of Web services for the automatic generation of rich user interfaces in order to facilitate physical mobile interaction with objects from the Internet of Things. We also present the results of an early user study and paper-prototyping in order to evaluate our approach to physical mobile interaction.

1 Introduction

Mobile phones have established themselves as devices for accessing and using digital information and services through the interaction with physical objects in the real world. Despite some technical constraints they are rich clients for mobile information access and capture. They are the most widely adapted personal computing platform, always connected to the Web, offer a wide area of technologies and modalities and are used in a rich, dynamic and personalized context.

The richer the means for information access become, the more information can be captured from the physical world and the more interactions with its objects become possible. These objects can be associated with digital information and services that go beyond their inherent amount of data. For example users can take pictures of visual markers [1] in magazines or on posters and use this information for the automatic invocation of associated services [2]. In combination with technologies like RFID or Near Field Communication (NFC) [3] physical mobile interaction [4] is increasingly gaining importance. It reduces mobile payment, identification or access control to simply swiping a mobile phone over a reader. NTT DoCoMo's i-mode FeliCa service

for example combines mobile phones with built-in NFC-chips and a service framework based on i-mode [5].

This development is greatly advanced by an increasing industrial effort to tag everyday objects with contactless RFID markers which facilitates their automatic identification, tracking, monitoring and recognition in manufacturing, transportation and logistics. That way, physical objects get individual digital identities, which also makes it easier to reference and present them on a network. This ultimately leads to an "Internet of Things" [6] in which objects as well as their associated information and services can be easily identified and accessed.

In our approach we want to improve physical mobile interaction with objects from the Internet of Things and transfer the familiarity of interacting with them to the interaction with associated information and services in order to make it easier and more intuitive. So far there are mostly simple solutions with single - often visual - markers that act as entry-points for mobile service interaction (see [1, 2]). We want to support more complex physical mobile interactions and shift their focus from mobile phones to physical objects. We want to push service functionalities and options off mobile phones, map them to multiple markers on physical objects and thus turn these into rich ubiquitous interfaces for new and more complex interaction techniques. Instead of struggling through glutted menus, users should be able to choose options and invoke services simply by touching appropriate wireless markers on physical objects.

We also think that interaction with objects from the Internet of Things can greatly benefit from the combination of physical mobile interaction and Semantic Web services. Therefore another focus of our research is to provide a framework that enhances the flexibility and expressiveness of Web services by applying Semantic Web technologies and thus facilitates the automatic generation of user interfaces for advanced physical mobile interaction from Semantic Web service descriptions.

The following chapters will evaluate research related to our approach (chapter 2), provide further details about our architecture (chapter 3) and present results of an early prototyping and user study (chapter 4). Chapter 5 will conclude this paper.

2 Background and Related Work

Fig. 1 shows an early draft of our architecture and outlines some of the key-issues we want to address within our research. Among them are the descriptions of Web services using Semantic Web technologies, the derivation of abstract user interfaces from them and the automatic generation of concrete mobile user interfaces that support the mobile interaction with physical objects? Context information is seen as an important influence on both interface generation and physical interaction.

We model the service infrastructure of our approach exploiting the flexibility of Web services as they provide services through a standardized interface and thus enable interoperability between heterogeneous applications and platforms. Web services are described by the Web Service Description Language (WSDL) [7] which specifies XML grammars for defining communication endpoints for message exchange between heterogeneous systems. Web services grounded on WSDL only provide descriptions and invocation details of single service endpoints. Therefore WSDL de-

scriptions are restricted to simple processes unless assisted by additional technologies for workflow description like OASIS's Business Process Execution Language for Web Services (BPEL4WS) [8] or W3C's Web Services Choreography Description Language (WS-CDL) [9]. Moreover WSDL is only capable of describing input and output messages syntactically. As a consequence Web services have to be invoked and connected manually as no mutual semantic relation can be described through WSDL.

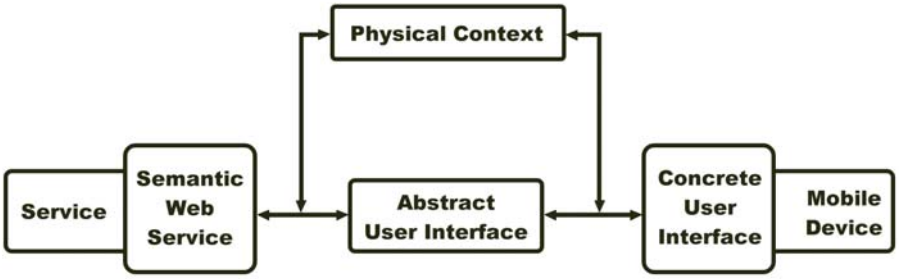


Fig. 1. Early draft of an architecture for physical mobile interaction with real world objects.

To address the shortcomings of WSDL, the Web Ontology Language for Services (OWL-S) [10] for Semantic Web services was developed. OWL-S is an ontology for describing Web services expressed in W3C's Web Ontology Language (OWL) [11]. OWL provides high expressiveness to enrich Web Services semantically. It enables automation of tasks related to Web services, among them capability-based discovery, execution, composition, interoperation, monitoring and recovery. Especially the automatic composition of multiple Web services is a main requirement for providing flexible interaction among them.

We evaluated several interface description languages as a basis for our own automatic interface generation: The User Interface Markup Language (UIML) [12] is a XML-based language for describing interfaces on an abstract and independent level. It supports the rendering of interface descriptions to different target languages using appropriate rendering engines but does not support the creation of several different interfaces from one single description. The Extensible Application Markup Language (XAML) [13] and the XML User Interface Language (XUL) [14] are both XML-based user interface description languages which provide a rich set of concrete graphical elements and widgets but no means for abstract interface description.

Since we include Semantic Web services in our architecture the most interesting approach to describe and generate interfaces is presented in [15]. Therein, OWL-S service descriptions are extended with user interface annotations modelled as OWL ontologies. These extended descriptions are used for the automatic generation and personalisation of form-based interfaces that allow the invocation of according Semantic Web services. The suggested system promises to be most useful for our own approach to generating dynamic interfaces from Web service descriptions.

Since we want to automatically and seamlessly link physical objects and information through the Internet, we also draw inspiration from an infrastructure for weaving

the Internet of Things that is currently maintained by EPCglobal Inc. [16]: Similar to our NFC-based approach this infrastructure relies on RFID-tags to attach a unique Electronic Product Code (EPC) to objects. The Object Naming Service (ONS) is used to match the EPC of an object with the URL of its associated information that is stored on a server. The Physical Markup Language (PML) [17] was developed to describe and store the information about an object on the network. PML descriptions of objects – whether provided by Web services or directly by NFC-tags - could be used as input for the invocation of associated Web services.

Our approach to physical mobile interaction is also inspired by similar research about closing the gap between the physical and the digital world: HPLab's Cooltown project [18] tries to link these two domains by providing people, places and things with an individual "web presence", which is similar to uniquely marking objects for easier network reference. [19] presents an early and basic approach to augmenting everyday objects like books, documents or business cards with RFID-Tags. The featured system uses the contactless identification of these objects to reference associated information and invoke simple associated actions on a tablet computer, e.g. opening the electronic file of a document when touching the tagged printout or displaying a person's home page after reading the tag on that person's business card. Since we want to provide more complex physical mobile interaction, our approach will be closer to [20] which also presents a framework for requesting services by touching RFID tags, including a middleware and tags with different functions.

3 Architecture

Our approach tries to combine two domains in order to realize physical mobile interaction: the Internet of Things and Semantic Web services. To bridge the gap between these two domains a mobile device acts as a generic mediator. Fig. 2 shows a high level architecture of our concept including an overview of employed technologies. In order to facilitate our implementation of physical mobile interactions and abstract their complexity we use the Physical Mobile Interaction Framework (PMIF) [4] which supports the interaction with visual markers, RFID/NFC, location information and Bluetooth.

We use the concept of a single Universal Client that is executed on a mobile device and manages the interaction with both physical objects and Web services. It is comprised of an Interaction Client component and a Service Client component which both interact with their corresponding domain.

The Interaction Client handles connections to physical objects and reads information – formatted e.g. with PML [17] - stored on their markers. It also provides one part of the user interface for the interaction with tagged physical objects. Depending on the concrete implementation it generates or at least displays parts of this interface from the interface description which it receives from the Interaction Proxy. The real world object can be seen as a physical extension of that interface. Its tags and markers act like options that hold information e.g. about unique identifiers or parameters for service invocation. That way the interface for interacting with services is spread on both the mobile phone and the physical object, e.g. a poster. This approach is flexible

enough to distribute different elements of the interface between mobile phone and physical object, depending on device capabilities, the number of usable tags and the complexity of the interface and its widgets.

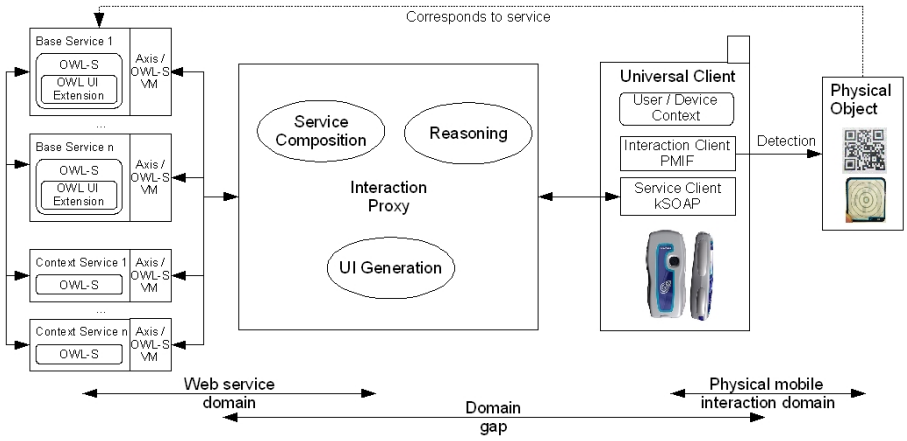


Fig. 2. High Level Architecture

Additionally the Universal Client could maintain context information about the user and the device which may improve automatic user interface generation and reasoning. For example the Universal Client could store a standardized user agent profile which may be used for generating the user interface with regard to the device capabilities.

The Service Client on the other hand provides a generic way of accessing services and controls the interoperation between the client and the service domain.

In the service domain we differentiate between Base Services and Context Services. All services are modelled as Semantic Web services. A physical object is associated with a single Base Service. Base Services are the main service components whereas Context Services provide context information for other services or the Universal client. Services can request each other to fulfil their tasks. By leveraging Semantic Web services we may ensure interoperability between different participating services independent from their underlying implementation or platform. Traditional Web services are enhanced by the support of automatic service selection, service invocation and service composition. Beside the service description, Base Services also provide ontological user interface annotations to enrich the generation of service user interfaces. We want to use OWL-S [10] ontologies to describe services and facilitate the automatic composition of multiple Web services. Moreover OWL-S provides automatic capability-based search which allows flexible and dynamic combination of services without the need of hard-wiring. We tend to use the OWL-S Virtual Machine [21] on top of the Apache Axis framework [22] to invoke and compose services.

As depicted in Fig. 2 we suggest a concept called Interaction Proxy which includes three main tasks for arranging the interoperation between the Universal Client and different Web services: service composition, reasoning and automatic user interface generation. When several different services are interconnected with physical objects, a composition of the services is required. Reasoning supports the interoperability be-

tween services by correlating semantic concepts. For example if a service requires a certain input, it may be derived from information that is already available. The automatic generation of a user interface is a main requirement to assist the physical mobile interaction process. This generation must be unified for all involved services to provide a consistent user experience. An abstract representation of the user interface may be derived from the service descriptions to a certain extent, including service inputs and outputs. However it must be mapped to a concrete representation to fulfil usability requirements. We picked up the idea of ontological user interface annotations described in [14] providing the mapping to a concrete user interface which can be rendered on the client. Describing the mapping as ontologies allows the reuse of reasoning facilities provided by the Interaction Proxy.

A central issue of our further investigations regarding the architecture will be the user's role during the interaction process. As the physical mobile interaction with services is a process with possibly several steps of execution a focus lies on the user's role in the interaction loop. This involvement calls for support for special cases such as reversible actions and fault actions.

4 Low Fidelity Prototyping and User Study

For our approach to mobile physical interaction we came up with two use case scenarios for mobile ticketing and conducted an early user study with low fidelity paper-prototypes of a mobile client application (see Fig. 3). The focus of our evaluation was the design and acceptance of our approach to mobile interaction with physical objects.

Fig. 3. Paper-prototype for physical mobile interaction

Fig. 4. Poster with NFC-tags

Fig. 5. NFC-enabled mobile phone

For our application we augmented 2 posters with Near Field Communication (NFC) [4] tags (see Fig. 4). Similar to pushing buttons on an automat, users can touch different options on the posters with an NFC-enabled mobile phone such as the Nokia 3320 with the Nokia NFC shell [23]. The mobile phone selects options through the recognition of the corresponding NFC-tags (see Fig. 5) that are attached to the back of the posters (see Fig. 4). After the user has assembled all necessary options, a client appli-

cation on the mobile phone would call a Semantic Web service associated with the poster (see Fig. 2).

This scenario is an example for an interface that is distributed between mobile phone and physical object: The mobile phone interface that is rendered from an interface description guides the interaction with markers on the physical object and manages the invocation of the associated Web service. The parameter-values for this invocation are stored on the NFC-tags which represent external options of the mobile phone user interface.

Fig. 6 shows the two posters we designed for our use case scenarios: The first poster allows the purchase of movie tickets and displays a number of appropriate options (title, cinema, etc.). The second poster implements a simplified way to buy tickets for the Munich transportation system. Instead of having to understand the complicated ticketing system, users only have to touch the station they want to start their journey from, their destination, the number of persons and the duration of the ticket. Unfortunately there are too many stations too close together on the map to put a different tag behind each station. Instead we tagged the 4 areas (white, green, yellow and red – see Fig. 6) the transportation map is divided into. Users would have to find their station of choice, remember its association to an area and touch the according tag.

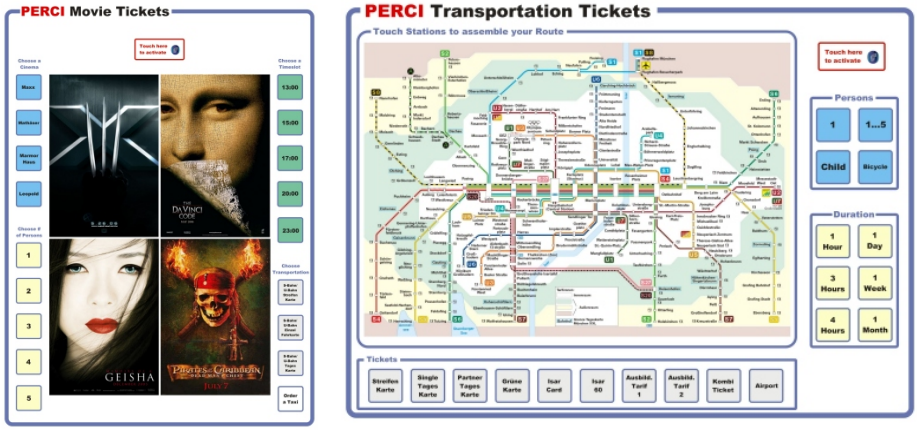


Fig. 6. Prototypes of posters for buying movie- (left) or transportation-tickets (right) [1][2][3][4][5]

We put up life-size versions of our posters in a corridor of the Institute of Media Informatics in Munich and conducted a survey with 10 participants. After an introduction on how NFC-based communication between an augmented poster and a mobile phone would basically work, the participants had to accomplish two tasks – buying

[1] X-Men 3 Poster. www.actuacine.net/Poster/x-3.jpg. © 2005 Twentieth Century Fox
[2] The DaVinci Code Poster. www.actuacine.net/Poster/davinci.jpg. © Columbia Pictures
[3] Memoires of a Geisha Poster. www.actuacine.net/Poster/geisha.jpg. © Sony Pictures
[4] Pirates of the Caribbean, Dead Man's Chest Poster. www.actuacine.net/Poster/pirates22.jpg. © Disney Pictures
[5] MVV Schnellbahn-Netzplan. http://www.mvg-mobil.de/pdf-dateien/netzplaene/schnellbahnnetzplan.pdf

tickets for a movie and transportation - using the posters and paper-prototype mobile phones (see Fig. 3). These included a mock-up of a client application that provided the users with feedback on their physical interactions with the poster and the associated services. Before and after the two tasks participants of the survey were asked a number of questions about the application and the scenarios.

The results of the survey can be summarized as follows: 75 % of the participants thought that the application was useful and would also use it if it was available – depending on the quality of the implementation and the number of available options. It was often mentioned that such a system could easily replace other automats (e. g. ticketing machines) but also that the human contact and individual feedback was lost compared to e.g. buying tickets at a counter. Although people without a technical background might at first have a certain inhibition level, they are considered to understand and successfully use the application – provided that they are familiar with a mobile phone. The system is largely considered to be intuitive and easy to use even when a certain initial effort is needed to understand it and to get accommodated to it.

When asked about advantages of the system the participants answered that it is faster and cheaper than regular automats and would cause less queuing. The posters can be put up everywhere and are easy to replace. Users are not bound to their desktop computer for ordering tickets online but can do it spontaneously using their mobile phones. The physical interaction is considered to generate fewer faults, to be easy and intuitive to use and to provide an additional value to the omnipresent mobile phone, especially in combination with mobile payment. The new application also causes technical sensation and interest, is considered to be visually interesting and attractive and provides a certain factor of fun. The distributed interaction between posters and mobile phones raises new possibilities for designing this interaction, pushing options from mobile phone menus onto posters and making them less complicated.

The given disadvantages of the system can be related to the system in general and to our concrete application and its lacks of design: For some participants the system was not intuitive enough to use. Posters can only be operated with mobile phones (alternatives should be provided) and have to be put up and actualised. The use of NFC-technology is mostly unknown and would have to be established for common use and interaction. Frequently switching the focus of user attention between the poster and the mobile phone was also seen as a disadvantage.

Many users asked for additional information, e.g. an introduction the system, more feedback during the interaction or a context-aware help for explaining options and the posters' workflow. Users also wanted to know what happened with their tickets after they bought them and how their possession of them was realized. Some users mentioned that a general handling of errors should be provided including the reversibility of actions, e.g. if users want to correct their selections.

As for the transportation poster, most users did not approve the employed method of choosing stations by associating them with different zones. At first most users intuitively touched the appropriate stations to select them. As no feedback was provided from the client, they looked for other means of selection and mostly found the options for the different zones of the transportation map. This definitely raised the need to find a new way for a more intuitive selection of stations.

Another interesting issue was the activator-tag that we placed on one poster and which prompted potential users to start interacting with the poster by touching it. We

considered it to be an explicit starting-point for the interaction and most users understood this concept. Nevertheless only half of them appreciated its intention as an explicit entry-point that enhances the structuring of the interaction and the feeling of controlling it. The other half of the users considered it to be useless and wanted to start the physical interaction implicitly by touching the first option on the poster. Sometimes it was even not clear, what was activated by touching this tag, whether it was the poster itself, the mobile client or just the tags.

5 Conclusion

We introduced our approach to mobile interaction with Web services through the physical interaction with associated real world objects. In our suggested architecture we try to exploit the enhanced expressiveness of Semantic Web service descriptions for the automatic generation of mobile interfaces to support this physical mobile interaction with objects from the Internet of Things. We are currently building a prototype for an entertainment use case scenario and conducted a survey using paper-prototyping to investigate the design and acceptance of our approach. As we get some affirmative answers out of this survey, it also shows lacks of our approach. We will have to redesign some of the widgets for physical interaction in order to make their application more obvious to users. Another important issue to think about will be how to guide users through our system and give them more feedback and hints about available options and actions.

Our next steps will cope with implementations on several levels in order to combine Semantic Web services, interface generation and physical mobile interaction in our proposed architecture.

6 Acknowledgement

This work was performed in the context of the Perci (PERvasive ServiCe Interaction) project [24] which is funded by DoCoMo Euro-Labs and the research project Embedded Interaction which was founded by the DFG (’Deutsche Forschungsgemeinschaft’).

References

1. Rohs, M., Gfeller, B.: Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects. In: Ferscha, A., Hoertner, H., Kotsis, G. (Eds.): Advances in Pervasive Computing, Austrian Computer Society (OCG), Vienna, Austria (2004). pp. 265-271
2. Rukzio, E., Schmidt, A., Hussmann, H.: Physical Posters as Gateways to Context-aware Services for Mobile Devices. Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004), English Lake District, UK (2004).
3. Near Field Communication, http://www.nfc-forum.org/

4. Rukzio, E., Wetzstein, S., Schmidt, A.: A Framework for Mobile Interactions with the Physical World. Invited paper special session "Simplification of user access to ubiquitous ICT services" at the Wireless Personal Multimedia Communication (WPMC'05) conference, Sept 18-22, 205 - Aalborg, Denmark.
5. NTT DoCoMo, iMode Felica. http://www.nttdocomo.com/corebiz/icw/index.html
6. Meloan, S.: Toward a Global "Internet of Things". November 2003. http://java.sun.com/developer/technicalArticles/Ecommerce/rfid/
7. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl
8. Business Process Execution Language for Web Services version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
9. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Barreto, C.: Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/
10. DAML Services Home Page. http://www.daml.org/services/owl-s/
11. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/
12. UIML.org. http://www.uiml.org/
13. Rector, B.: Introducing "Longhorn" for Developers. December 2003. http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnintlong/html/longhornch03.asp
14. XML User Interface Language (XUL) Project. http://www.mozilla.org/projects/xul/
15. Khushraj, D., Lassila, O.: Ontological Approach to Generating Personalized User Interfaces for Web Services, 4th International Semantic Web Conference (ISWC 2005), LNCS 3729, Springer-Verlag Berlin Heidelberg (2005). pp. 916–927
16. EPCglobal Inc. Home Page. http://www.epcglobalinc.org/
17. PML The Physical Markup Language. http://web.mit.edu/mecheng/pml/index.htm
18. Kindberg, T., Barton, J., Morgan, J., Becker, J., Bedner, I., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Pering, C., Schettino, J., Serra, B., and Spasojevic, M.: People, Places, Things: Web Presence for the Real World. In proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000). In MONET Vol. 7, No. 5 (October 2002).
19. Want, R., Fishkin, K.P., Gujar, A., Harrison, B.L.: Bridging Physical and Virtual Worlds with Electronic Tags. CHI 1999: 370-377
20. Riekki, J., Salminen, T., and Alakarppa, I. 2006. Requesting Pervasive Services by Touching RFID Tags. IEEE Pervasive Computing 5, 1 (Jan. 2006)
21. Srinivasan, N., Paolucci, M. OWL-S Virtual Machine. http://owl-s vm.projects.semwebcentral.org/
22. WebService – Axis. http://ws.apache.org/axis/
23. Nokia - Nokia NFC. http://www.nokia.com/nfc
24. Perci (PERvasive ServiCe Interaction) Homepage: http://www.hcilab.org/projects/perci/

Mobile Interaction with the Internet of Things

Sven Siorpaes[1], Gregor Broll[1], Massimo Paolucci[2], Enrico Rukzio[1],
John Hamard[2], Matthias Wagner[2], Albrecht Schmidt[1]

[1] Embedded Interaction Research Group,
Media Informatics Group, University of Munich
{sven, gregor, enrico, albrecht }@hcilab.org
[2] DoCoMo Eurolabs
{paolucci, hamard, wagner}@docomolab-euro.com

Abstract. The interaction between mobile devices and physical objects in the real world is gaining more and more attention as it provides a natural and intuitive way to request services associated with real world objects. We currently see several approaches for the provision of such services. Most of them are proprietary, designed for a special application area or interaction technique and provide no generic concept for the description of real world services. On the other hand the *Internet of Things* provides a set of standards and methods to tag objects in the real world. We think that the combination of these two technologies can support the development and dissemination of mobile interactions with the real world. Therefore, in this paper we present a concept, an architecture and an early prototype currently under development for mobile interactions with the Internet of Things. Hereby we use Semantic Web services for the description of services provided by the physical objects. This service description is then used for the automatic generation of user interfaces rendered by the mobile device.

1 Introduction and Motivation

Today users are more and more immersed into a complex sphere of ubiquitous information. Mobile clients offer increasingly sophisticated methods to capture information, to make use of context information and to interact directly with objects in the real world. On the other hand physical objects are increasingly associated with digital information through the augmentation with visual [1] and wireless markers such as RFID tags. In this context physical mobile interactions [2] allow users to select virtual information and invoke services through the interaction with objects in the real world [3, 4]. Currently there are several approaches for the provision of applications that take such interactions into account. Most of them are proprietary, designed for a special application area or interaction technique and provide no generic concept for the description of real world services. One example is the Nokia Local Interactions Server which is a real-time web service that acts as a back end for RFID-based mobile interactions [5].

In addition we currently see a big interest in industry and academia in the *Internet of Things* in which real world objects have an individual digital presence [6]. Here physical objects are uniquely identified and described in a standardised way which

facilitates access to and interaction with them. We think that the combination of physical mobile interactions and the Internet of Things can support the development and deployment of mobile interactions with the real world. Therefore, in this paper we present a concept, an architecture and an early prototype currently under development for mobile interactions with the Internet of Things, its objects and their associated services. Hereby we use Semantic Web services for the description of services associated to physical objects. This service description is then used for the automatic generation of a user interface on the mobile device.

Web service technology provides a new way of making information and services available while reducing interoperability issues and enhancing extensibility, platform independence and standardized exchange of messages. Furthermore we want to improve their flexibility and expressiveness by adding semantic descriptions and thus enhance the modelling and dynamic composition of web services. As shown in [7] modelling services as Semantic Web services is powerful enough to acquire implicit context information by composing web services and therefore relieves the user of providing information explicitly. Thus, we exploit in our architecture the expressiveness of Semantic Web services for the automated generation of user interfaces rendered by the mobile device.

We want to explore how such user interfaces can be optimized to provide easier and more familiar interaction with physical objects in the internet of things and the services associated with them. Since there is still no consistent way to integrate web services and means for physical interaction, our architecture has to meet several technical requirements. Among them are:

- Modelling, composition and provision of Semantic Web services;
- Description, automatic generation and integration of mobile user interfaces to abstract the complexity of web service functionalities and support the user interaction with physical objects;
- Connection between mobile devices and marker technologies as well as modelling and exchange of messages among components.

Several efforts are underway dealing with automatic generation of mobile user interfaces. The Pebbles project focuses on the interaction of mobile devices with physical appliances such as TV or VCR [8]. Although web service semantics provide only limited support for the description and creation of user interfaces, [9] explains how a service description extended by semantic user interface annotations is capable of automatically generating a user interface that is both highly flexible and expressive. We consider this approach as an interesting starting point for our own automatic user interface generation.

2 Architecture and Prototype

As mentioned, there are two separate domains which we want to combine in our approach: the Internet of Things and physical mobile interactions where mobile devices are used to interact with physical objects. The main goal of our approach is to connect these two domains whereas the mobile device acts as a mediator between them. Fig. 1 depicts a high level view of our architecture. The mobile device acts as a

Universal Client which is independent from the physical objects it interacts with and also from services it invokes. To interact with both domains it uses different components denoted as *Interaction Client* and *Service Client*. The Interaction Client detects unique identifiers and additional data stored on the *Physical Object* while the Service Client communicates with the service domain. The Universal Client stores user context information and device capabilities which could enrich the automatic user interface generation. As device context we consider several mobile platforms which vary in their physical interaction capabilities (e.g. camera or RFID/NFC reader) and user interface capabilities (e.g. XHTML browser or J2ME runtime environment). Therefore, the Universal Client has to be able to support an arbitrary combination of device capabilities involved in the interaction process.

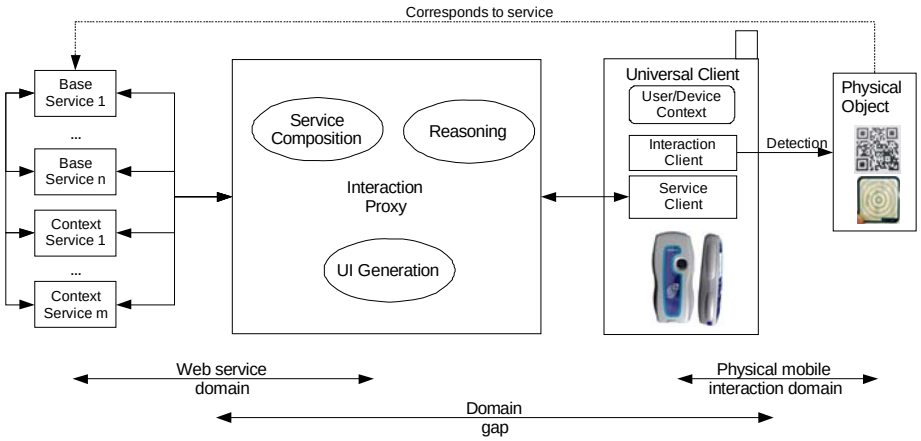


Fig. 1. High level architecture for physical mobile interactions with the Internet of Things

Services in the web service domain are described as Semantic Web services to facilitate interoperability, automatic service invocation/composition and automatic user interface generation. Services are grouped into *Base Services* and *Context Services*. Base Services execute the main functional tasks related to the interaction with physical objects. There may be a fixed relationship between a Physical Object and an initial Base Service. Context Services provide context information such as location and time. All services can interact with each other. For example a Base Service can request other Base Services or Context Services. However the Universal Client should also be capable of accessing Context Services directly.

For connecting the service domain and the physical interaction domain we propose a concept called *Interaction Proxy* which provides three main functions: service composition, reasoning and user interface generation. Service composition describes the interaction with several web services that can be involved in the physical interaction process. Reasoning is required to resolve the lack of semantic interoperability between different services.

Another focus of our work lies in the automatic generation of a user interface for different services which should be provided to the user in a consistent and transparent way. By providing a definition of required inputs and outputs for the service

invocation, the semantic service description already defines a raw structure of the device user interface. In our approach the semantic service description is enhanced by an additional user interface extension which describes a parameter type-based mapping to a concrete user interface.

From the architectural point of view we identified different approaches to which domain the Interaction Proxy can belong. The user's privacy could be ensured by assigning the Interaction Proxy to the Universal Client. On the other hand the process of reasoning is presumably too computationally demanding for mobile phones. Therefore we decided to have a hybrid approach in which the Interaction Proxy concept is split into a device and a server component.

To illustrate our concept we are currently implementing an early prototype. We have defined a mobile commerce scenario in which a poster from a movie distribution provider offers a ticketing service for different cinemas. We assume different movies being advertised on a poster which have to be explicitly selected by the user. As shown in Fig. 2 a Near Field Communication (NFC) [10] enabled mobile phone is used for the interaction with the movie poster. The NFC/RFID tags are fixed on the back of the poster as depicted in Fig. 3. The service can use implicit context information such as location to determine the nearest cinema or time to restrict the starting time slot of the movie. In this simple scenario there is a main movie distribution service composed of several different cinema services and context services.

Fig. 2. Physical mobile interaction between NFC equipped device and poster.

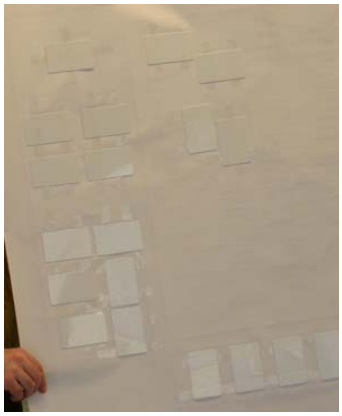


Fig 3. NFC tags fixed on the back of the poster.

The implementation of the framework and the interacting service components is currently under development. For describing Semantic Web Services we intend to use OWL and OWL-S [11] descriptions which can be developed with the ontology modelling tool Protégé [12] in combination with the Protégé OWL-S plug-in. Furthermore we plan to deploy Web Services with the Apache Axis framework. For invoking and composing multiple services we use the Mindswap OWL-S API [13] on top of Axis.

3 Conclusion

So far the vision of an Internet of Things is restricted to the standardized description of physical objects. Enhancing physical objects with service interaction support is still only accomplished by proprietary solutions. In this work we discussed the idea of combining physical mobile interactions and the Internet of Things in a generic way. We presented a system enabling the mediation between physical objects and multiple services through a *Universal Client*. Our work focuses on the composition of independent services which should be provided to the user in a consistent and seamless way. By using Semantic Web service technologies we see a great chance to overcome the semantic incompatibility between different services. Moreover we can benefit from describing services semantically to automatically generate a uniform user interface utilizing the proposed semantic user interface annotations. A prototype based on an entertainment scenario is currently under development. The next steps will consist in the evaluation of our concept and the improvement of our implementation following an iterative design process.

4 Acknowledgement

This work was performed in the context of the PERCI (PERvasive ServiCe Interaction) project [14] which is funded by NTT DoCoMo Euro-Labs and the research project Embedded Interaction which is founded by the DFG (‘Deutsche Forschungsgemeinschaft’).

References

1. Michael Rohs, Beat Gfeller. Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects. In: Alois Ferscha, Horst Hoertner, Gabriele Kotsis (Eds.): Advances in Pervasive Computing, Austrian Computer Society (OCG), ISBN 3-85403-176-9, pp. 265-271, Vienna, Austria, April 2004
2. Rukzio, E., Wetzstein, S., Schmidt, A.: A Framework for Mobile Interactions with the Physical World. Invited paper special session "Simplification of user access to ubiquitous ICT services" at the Wireless Personal Multimedia Communication (WPMC'05) conference, Sept 18-22, 205 - Aalborg, Denmark.
3. Enrico Rukzio, Albrecht Schmidt, Heinrich Hussmann. Physical Posters as Gateways to Context-aware Services for Mobile Devices. Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004), English Lake District, UK, 2004.
4. Jukka Riekki, Timo Salminen, Simo Hosio, Ismo Alakärppä. Requesting services by touching objects in the environment. In Proceedings of the 11th International Conference on Human-Computer Interaction, Las Vegas, NE, 2005.
5. Nokia Local Interaction Server. http://www.europe.nokia.com/nokia/0,,76300,00.html.
6. Steve Meloan. Toward a Global “Internet of Things”. In: Sun Developer Network. 2003. http://java.sun.com/developer/technicalArticles/Ecommerce/rfid/
7. Mithun Sheshagiri, Norman M. Sadeh, Fabien Gandon. Using Semantic Web Services for Context-Aware Mobile Applications. MobiSys 2004 Workshop on Context Awareness, Boston, 2004.

8. Jeffrey Nichols. Automatically Generating User Interfaces for Appliances. In: Ferscha, A., Hortner, H., Kotsis, G. (eds.): Advances of Pervasive Computing, pp. 105-110, 2004.
9. Deepali Khushraj, Ora Lassila. Ontological Approach to Generating Personalized User Interfaces for Web Services. In Gil, Y., Motta, E., Benjamins, V. R. (eds.): The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, number 3729 in Lecture Notes in Computer Science. Springer-Verlag, Galway (2005) 916-927
10. Near Field Communication, http://www.nfc-forum.org/home
11. DAML Services: http://www.daml.org/services/owl-s/
12. The Protégé Ontology Editor and Knowledge Acquisition System: http://protege.stanford.edu/
13. Mindswap OWL-S API: http://www.mindswap.org/2004/owl-s/api/
14. Perci: http://www.hcilab.org/projects/perci/index.htm

Ablauf

- Zur Einleitung die Funktionsweise des Posters und der NFC-Tags kurz erklären, um das System allgemein verständlich zu machen.
- Client auf dem Handy, mit dem Tags auf einem Poster berührt und „angeklickt“ werden können. Durch dieses Anklicken können verschiedene Optionen ausgewählt werden, um verschiedene Aufgaben erfüllen zu können.
- Kurze Demo mit dem Nokia NFC-Handy; Handy auf beliebiges Tag halten um zu zeigen, wie dieses vom Handy erkannt wird.
- Hinweis zum Ablauf des Prototyping
    - Vorherfragebogen
    - Durchspielen des eigentlichen Szenarios
    - Nachherfragebogen
- Vorherfragebogen abfragen, Antworten protokollieren
- Szenario 1; Beschreibung zum Kino-Poster vorlegen, passende Paper-Handys vorlegen und Verhalten des Benutzers gegebenenfalls protokollieren
- Szenario 2; Beschreibung zum MVV-Poster vorlegen, passende Paper-Handys vorlegen und Verhalten des Benutzers gegebenenfalls protokollieren
- Nachherfragebogen abfragen, Antworten protokollieren

Vorher-Fragebogen

1. Würdest Du ein solches System nutzen wollen? (Ja/Nein)

2. Wie schätzt Du die Nützlichkeit des vorgestellten Systems ein? Bewertung nach Schulnoten (+ Kommentare)

3. Welche Vorteile könnte das System Deiner Meinung nach haben?

4. Welche Nachteile könnte das System Deiner Meinung nach haben?

5. Eignet sich ein solches System Deiner Meinung nach auch für Leute ohne technischen Hintergrund? Wie einfach glaubst Du, ist das System zu bedienen?

Szenario 1 (Kinokarten bestellen)

Du möchtest mit Deinen Freunden spontan ins Kino gehen und stehst zufällig vor einem NFC-Kinoplakat. Benutze den Perci-Client auf Deinem NFC-fähigen Handy, um mit Hilfe dieses Posters

- 3 Karten
- für die Nachmittagsvorstellung
- des Films „Die Geisha“
- im „Marmorhaus“

zu bestellen.

Szenario 2 (MVV-Ticket kaufen)

Nachdem Du die Karten fürs Kino gekauft hast, brauchst Du noch eine Möglichkeit, um in die Innenstadt von München zu kommen. Benutze wiederum den Perci-Client auf Deinem NFC-fähigen Handy, um mit Hilfe des nächsten Posters eine MVV-Karte für Dich und Deine beiden Freunde zu erwerben.
Für die Wahl Deiner Karte gib bitte an, dass

- Ihr insgesamt 3 Personen seid,
- die von Großhelfendorf zur Münchener Freiheit fahren müssen
- und wahrscheinlich den ganzen Tag lang unterwegs sind.

Beachte diesmal bitte, dass Du als ersten Tag den Aktivator-Tag auf dem Poster anklicken sollst.

Protokoll (zu beiden Szenarien)

Nachher-Fragebogen

Würdest Du ein solches System nutzen wollen? (Ja/Nein)

Wie schätzt Du die Nützlichkeit des vorgestellten Systems ein? Bewertung nach Schulnoten (+ freie Antworten)

Welche Vorteile hat das System Deiner Meinung nach?

Welche Nachteile hat das System Deiner Meinung nach? Verbesserungsvorschläge?

War die allgemeine Bedienbarkeit des Systems intuitiv? Was könnte verbessert werden?

Genügten die gegebenen Hinweise, um das System bedienen zu können? Welche anderen Hinweise könnten nützlich sein?

Eignet sich ein solches System Deiner Meinung nach auch für Leute ohne technischen Hintergrund? Wie einfach glaubst Du, ist das System zu bedienen?

Ist der Aktivator-Tag Deiner Meinung nach nützlich?

Welche anderen Szenarien könntest Du Dir für solche NFC-Poster vorstellen?

Fragen die die Benutzerstudie beantworten soll:

- Was ist besser (schneller, bevorzugt, etc.)?: Alles Interaktionen direkt am Handy (direct input) oder lieber direkt mit dem Poster interagieren (touching / pointing)?
- Vergleich der Interaktionstechniken.
- Wie lange schauen die Personen auf das Poster und auf das Handy? Wie lange brauchen sie um ein Ticket zu kaufen ?(kann durch die Auswertung des Videos herausgefunden werden),

Perci - Evaluation

Demographie

Geschlecht ○ weiblich ○ männlich

Alter _____

Arbeit

○ ich bin Student: Welches Fach: _____

○ Beruf Welcher: _____

I. Explaining the Protoype

Just one poster is used.
Explaining (just explaining) the prototype and the three interaction techniques (Touching – NFC, Pointing – Marker, direct input)

II. Interaction techniques

Wie schätzt du die drei Interaktionstechniken ein?

1 - Touching (NFC):

_____

_____

_____

_____

2- Pointing (Marker):

_____

_____

_____

_____

3- Nummerneingabe:

_____

_____

_____

_____

Welche Interaktionstechnik würdest du bevorzugen (nur eine darf genannt werden) und warum?

_____

_____

Welche Interaktionstechnik ist deiner Meinung nach am schnellsten (nur eine darf genannt werden) und warum?

_____

_____

III. Usage of the protoype

Every interaction technique should be used.
Die Reihenfolge der zu testenden Interaktionsform wird durch einen Zufallsgenerator (http://www.agitos.de/zufallsgenerator.php, Würfel) bestimmt. So kann z.B. folgende Reihenfolge auftreten: 3, 1, 2 (1-Touching, 2-Pointing, 3-Direct Input)
Wir geben nur wenig Hilfe bei der Bedienung, nur wenn es gar nicht mehr anders geht, um zu sehen wie die Zielperson mit dem Prototyp / der Interaktionstechnik zurechtkommt.

Errechnete / Verwendete Reihenfolge der Interaktionstechniken:

Die Zielperson soll laut denken.

Touching (NFC)

_____

_____

_____

_____

_____

Pointing (Marker)

_____

_____

_____

_____

_____

Direct Input

_____

_____

_____

_____

_____

IV. Nach der Benutzung des Prototypes

Die Reihenfolge der Befragung zu den Interaktionstechniken soll auch in einer gleichverteilten Reihenfolge (http://www.agitos.de/zufallsgenerator.php) erfolgen.

Touching (NFC)

	trifft voll zu	**trifft eher zu**	**Weiß nicht**	**trifft weniger zu**	**trifft gar nicht zu**	**Kommentare**
Einfache Handhabung	○	○	○	○	○	_____ _____ _____
Lustig	○	○	○	○	○	_____ _____ _____
Innovativ	○	○	○	○	○	_____ _____ _____
Zuverlässig	○	○	○	○	○	_____ _____ _____

Pointing (Marker)

	trifft voll zu	**trifft eher zu**	**Weiß nicht**	**trifft weniger zu**	**trifft gar nicht zu**	**Kommentare**
Einfache Handhabung	○	○	○	○	○	_____ _____ _____
Lustig	○	○	○	○	○	_____ _____ _____
Innovativ	○	○	○	○	○	_____ _____ _____
Zuverlässig	○	○	○	○	○	_____ _____ _____

Direct Input

	trifft voll zu	trifft eher zu	Weiß nicht	trifft weniger zu	trifft gar nicht zu	Kommentare
Einfache Handhabung	○	○	○	○	○	_____ _____ _____
Lustig	○	○	○	○	○	_____ _____ _____
Innovativ	○	○	○	○	○	_____ _____ _____
Zuverlässig	○	○	○	○	○	_____ _____ _____

Welche Interaktionstechnik würdest du bevorzugen (nur eine darf genannt werden) und <u>warum</u>?

_____

_____

Welche Interaktionstechnik ist deiner Meinung nach am schnellsten (nur eine darf genannt werden) und <u>warum</u>?

_____

_____

V. Vergleich der Interaktionen

Ranking - Bevorzugte Interaktion?
Hier werden immer von 1-3 Plätze für die Interaktion vergeben.

	Touching (NFC)	**Pointing (Marker)**	**Nummerneingabe**
Bevorzugte Interaktion	____	____	____
Dauerhaft nutzen	____	____	____
Zuverlässigste Technologie	____	____	____
Innovativste Interaktion	____	____	____
Spaßigste Interaktion	____	____	____