ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Faculty of Engineering and Computer Science
Institute of Media Informatics

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Medieninformatik

Modality-independent Exchange of Information Across Devices Using the Pick-and-Drop Concept

Master thesis at the University of Ulm
Masterarbeit an der Universität Ulm

Submitted by/Vorgelegt von:
Michael Barth
michael.barth@uni-ulm.de

Examiner/Gutachter:
Prof. Dr. Michael Weber
Prof. Dr. Enrico Rukzio

Adviser/Betreuer:
Dipl.-Inf. Frank Honold

2013

„Modality-independent Exchange of Information Across Devices Using the Pick-and-Drop Concept“
Copy from September 13, 2013

ACKNOWLEDGEMENTS/DANKSAGUNGEN:

This work would not have been possible without the advice and support of many people.

First and foremost, I wish to express my gratitude to Prof. Dr. Michael Weber *and my adviser* Frank Honold *for providing me the opportunity for this project. I especially wish to thank my adsiver for offering his invaluable guidance throughout this project and for taking special care to keep me motivated during its stressful last stages. I am also grateful to* Prof. Dr. Enrico Rukzio *for offering his knowledge and guidance on the topic at the early stages.*

Special thanks go to the many research associates who sacrified their time to evaluate this work and enrich it with their valuable feedback.

I also like to express my gratitiude to Christian Seitzer *and* Miriam Klement *for proofreading my work and adding their honest opinions.*

My deepest thanks go to my family for their undivided support and affection. Thanks for always being there!

As usual, the best has been saved for last: I wish to thank the most precious person to me, Alexandra Klement*, for her continuous love, support and positive attitude that seems to be unquenchable. Also, thank you for finding the time to proofread my work as well!* ;-)

This work was created utilising free software
Bei der Erstellung dieser Masterarbeit wurde freie Software eingesetzt:

Typeset/Satz: PDF-LaTeX 2$_\varepsilon$
Printing/Druck: Kommunikations- und Informationszentrum (kiz), Universität Ulm

Contents

1 Introduction

This work serves as master thesis in the field of human-computer interaction. Its focus lies in the realisation of a modality-independent interaction concept based on the Pick-and-Drop concept [Rek97] in a model-based approach.

This chapter starts by stating the motivation behind the topic, elaborates on the problem this thesis aims to solve, describes the approach that has been taken and finally outlines the structure of this document.

1.1 Motivation

The diversity and ubiquity of computing devices that surround us is increasing steadily, as is the number of devices people use in their daily lifes [Wei95, Seb09]. There are dozens of different types of computational devices that come in diverse shapes, sizes and with a varying set of features. These devices range from classical desktop computers to laptops, netbooks, tablets, smartphones, (multi-)touch tables, high-definition televisions (HDTVs) and gaming consoles.

With the ever-growing diversity of devices, the input and output modalities also diversified. In human-computer interaction, the term modality can be described either as *"a sense through which the human can receive the output of the computer (for example, vision modality)"* or as *"a sensor or device through which the computer can receive the input from the human"* [Wik13e]. In short, the term describes a method of communication between a human and a computational device in the field of human-computer interaction. When talking about modalities, it is sometimes necessary to distinguish between the definition of humans receiving output from the computer, which will be called **output modalities**, and inputs that the computer receives from the human, which will be called **input modalities** henceforth.

Although keyboard and mouse are the dominant method of interaction with desktop computers, there exists a number of other input modalities that have prospered – especially on other devices. For example, touch screens and gestures are very popular nowadays on smartphones, but speech input is also slowly gaining more ground [Ram03, ZNK10]. Today's software has to adapt to and utilise these diversified device features and modalities to make the most use of this ubiquitous computing environment.

1.2 Problem Statement

Despite this manifold diversity of devices and features, interaction between the involved devices is most often limited and inconvenient. Even the simple exchange of information between two similar devices, like computers on the same desk, is unnecessarily cumbersome [Rek97], let alone transferring information between fundamentally different devices.

Furthermore, a straight exchange of information is more constrictive than it needs to be and not always exactly what fits the user's needs. For example, different device features may necessitate a change of the information's representation. Like when the device features required for a certain form of rendering are not available – e.g. speakers for an audio file. Other times, using another type of modality might increase usability or be more practical for the intended use of the information – e.g. when following a manual, it is more useful to hear the instructions while following them instead of interrupting the task at hand every time to read and memorise the next step, then execute it.

To fully utilise all the possibilities of a ubiquitous computing environment, the interactions with and the information stored on computational devices have to become more flexible and adaptable. This applies to the presentation of the information as well. A smoother, more comfortable interaction has to focuses on the user's task, not on the limits and methods of interaction enforced by the devices, the software or the representation of the information.

The following work is based on an existing prototypical system that utilises a model-based approach to user interface generation at runtime. This allows the flexible and dynamic adaption of the user interface to the context of use. To our knowledge, a drag-&-drop-like interaction has never been adapted to a model-based user interface, allowing the content to adapt to the context of use as well.

1.3 Objective

This work aims to develop an interaction concept that enables an easy and spontaneous exchange of information across multiple devices, allowing the user to freely switch between input as well as output modalities, while performing a drag-&-drop-like interaction with the information objects. Furthermore, a prototypical implementation of the interaction concept that extends an existing system will be developed and evaluated to gain insight and feedback on the feasibility and usability of the proposed interaction concept, to learn what works well and what can be improved. The drag-&-drop-like interaction will be adapted to the model-based approach of the existing system.

1.4 Approach

First, a research on the current state of the art on available input & output modalities and methods of interaction is done to get a better understanding of today's possibilities and limits. Afterwards, an interaction concept for the exchange of information across device borders, inspired by Rekimotos Pick-and-Drop [Rek97], is developed that utilises the most effective and proven interaction methods for all prevalent modalities in use nowadays. The chosen methods of interaction represent the best practices and insight gained in other works. It is a fusion of the state of the art to offer the most fitting ways to realise a drag-&-drop-like interaction for a model-based multimodal user interface.

The core of the interaction concept will be the conversion from concrete, modality-specific representations of information to an abstract, modality-independent information and vice versa. This conversion is applied when picking an information object up on one device, using one specific modality available on that device, and then dropping it onto another device, potentially utilising another modality. As an example, picking up a book on an eBook reader and dropping it on a TV screen to watch the corresponding movie. This will be illustrated using scenarios showing possible use cases for this concept. To demonstrate this and gain some actual feedback on the concept, a prototype is implemented and evaluated.

1.5 Outline

This work is structured into six chapters. The first chapter introduces the reader to this work, outlining it by stating the motivation, problem and objective. Following this, the fundamentals upon which this work is built are briefly introduced in chapter 2. They are divided into two broad fields: Interaction fundamentals and multimodal user interfaces. The existing system is briefly introduced as an example of a multimodal user interface at the end of chapter 2.

Having the basics covered, the main body is started by chapter 3 which describes the interaction concept on an abstract and concrete level in theory as well as explaining some identified problems. Chapter 4 details the prototypical implementation, illustrating its architecture and integration into the existing system. Finally, an expert evaluation that has been conducted is elaborated in chapter 5.

Concluding, a summary of this work is given in chapter 6 with suggestions for future work.

2 Fundamentals

This chapter aims to introduce the two fundamentals upon which this work is based: Interaction fundamentals and multimodal user interfaces. Many different disciplines are combined under the term ‘interaction’ in the context of computational devices. They are described in the first section of the chapter at hand. The second section elaborates on multimodal user interfaces, their typical architectures, the current state of the art in this field, design principles, and finally introduces the existing (multimodal) system which this work extends.

2.1 Interaction Fundamentals

This section summarises the history of human-computer interaction (HCI), common input and output concepts necessary to enable any interaction with computers at all, common interaction concepts that developed over time, like drag & drop, and how all of this affects the coming age of ubiquitous computing.

2.1.1 Human-Computer Interaction

The concepts of HCI were, most of the time, first developed at universities and then refined by corporate research before they made their way into consumer products. This includes the concept of *direct manipulation of graphical objects* using a *pointing device*, which was first demonstrated by Ivan Sutherland in his doctoral dissertation in 1963. [Mye98]

Later in 1968, a research project at the Massachusetts Institute of Technology (MIT) featured *iconic representations* and *gesture recognition*. Using these concepts they realised dynamic menus with item selection utilising a pointing device and selection of icons by pointing. David Canfield Smith coined the term “icon” later in his doctoral thesis in 1975 and popularised them as one of the chief designers of the Xerox Star. [Mye98]

Today's computer systems are unthinkable without these basic concepts and they have been used more and more in the computer systems of the last 20 years. *Graphical user interfaces* (GUIs) that feature directly manipulable graphical objects and overlapping windows, which have been first proposed by Alan Kay in his 1969 doctoral thesis [Mye98], are the de facto standard currently.

The first *trainable gesture recognition* was realised by Warren Teitelman in 1964. At this time, it was common for light-pen-based systems to feature gesture recognition. It also has been used in commercial CAD systems since the 1970s, came to common notice with the Apple Newton in 1992, and is heavily utilised with touch-based technologies like smartphones nowadays.

The earliest approaches to *automatic speech recognition* (ASR) have been made 1967 and were based on finding speech sounds and providing appropriate labels to these sounds. Later in 1975, the first pattern matching approach was developed, which introduced the idea of pattern training. According to [AK10], *"the pattern-matching approach has become the predominant method for speech recognition in the last six decades"*. Recently, ASR has become more commonly used in consumer products with the introduction of Siri on the Apple iPhone 4S. Siri offers a natural language UI *"to answer questions, make recommendations, and perform actions by delegating requests to a set of Web services"* [Wik13f].

A quote from Brad Myers [Mye98] shall conclude this section: *"As computers perform faster, more of the processing power is being devoted to the user interface. The interfaces of the future will use gesture recognition, speech recognition, "intelligent agents", adaptive interfaces, video, and many other technologies now being investigated by research groups at universities and corporate labs. Therefore, it is imperative that university, corporate, and government-supported research continue and be well-supported, so that we can develop the science and technology needed for the user interfaces of the future."*

2.1.2 Common Input & Output Concepts

With the proliferation of computational devices and the advances made in human-computer interaction, the input and output concepts have also diversified. The most common input and output concepts will be introduced in this section. This is important to make appropriate choices in HCI design. [ZNK10]

Input Concepts

Keyboard and mouse are still one of the most prevalent input concepts, being used to operate most computers today – be it desktop machines or laptops. For mobile and ubiquitous computing, however, they represent rather constraining and limited interaction concepts [SDD12]. With the progress made on touch screens, touch input has mostly replaced pen-based input on mobile devices and is more popular nowadays, since almost all smartphones and tablets come with touch screens.

Gestures have become popular with the proliferation of touch input. They are mostly used in conjunction with touch input as it feels more natural to use them with fingers than to use them with a mouse [SDD12]. The Microsoft Kinect popularised motion sensing and body gestures in video games, but is mostly used in research otherwise. ASR is utilised more on mobile devices than on desktop PCs, with Apple's Siri being the prime example. Although operating systems like Windows also support speech input, it is seldom used there.

In the remainder, this work will consider the mouse, keyboard, touch, gestures (subdivided into mouse and touch gestures) and ASR as potential input concepts.

Output Concepts

The traditional five senses of the human organism are sight, hearing, smelling, tasting and touching [Seb09]. Smelling and tasting have rather limited uses in human-computer interaction and are thus not considered. Tactile or haptic feedback is used in some specialised cases, like vibrations on smartphones or force feedback joysticks in games. Due to its limited, specialised nature it is also not considered in this work.

The visual channel is one of the most powerful senses to receive and transfer information, as the saying *"A picture is worth a thousand words"* already implies. Text and pictures are examples of the media that can be perceived through the visual channel. It is especially strong at communicating spatial information. The auditive channel can be used to communicate information through language and meaning through sound. Humans naturally relate certain things with specific sounds, like the *ding-dong* sound a doorbell makes or the well-known sound when an error occurs in Windows. Such sounds are sometimes called auditory icons or earcons. Hearing can also be used to infer spatial information or direction, but this is more difficult than using the visual channel.

Together, the visual and aural channel form the most powerful combination to communicate information to a user. Videos, a combination of pictures with audio, communicate information on two different channels at the same time, combining meaning with spatial information. These two output concepts are considered in this work for system feedback.

2.1.3 Related Interaction Concepts

Throughout the history of HCI, some common concepts were developed that are used on many platforms and operating systems, like the cut/copy and paste operations, the clipboard, drag & drop, or gestures. There is also a lesser-known specialisation of drag & drop called Pick-and-Drop [Rek97]. These concepts are introduced in the following subsections as they are related to this work.

Cut, Copy, and Paste

Cut, copy, and paste are interface metaphors borrowed from manuscript editing, where text was edited by cutting it with a scissor and pasting it on a paper to create a page layout. In computer systems, *copy and paste* or *cut and paste* are operations used to transfer text, data, files or objects from a source to a destination. For cut and paste operations, the data is removed from its source location and moved to the destination, whereas copy creates a duplicate of the data at the destination. [Wik13b]

To allow this, a clipboard is used as temporary data storage. This interaction technique is closely related to GUIs and drag & drop. [Wik13b]

The Clipboard

The clipboard is a short-term data storage that is used to transfer data between locations and applications on most computational devices. This is done by utilising the *cut, copy and paste operations*. The clipboard is mostly used in conjunction with GUI applications and implemented as a hidden temporary data buffer. [Wik13a]

It uses the natural metaphor of a clipboard where you can temporarily clip on documents and remove them later on.

Drag & Drop

In GUIs, drag & drop is the concept of "grabbing" a virtual object and *dragging* it to a new location where it will be *dropped*. This is done using a pointing device and is an example of direct manipulation of a graphical object (the icon most of the time). [Wik13c]

The dragged object can be dropped in the same window, a different window, in an application or on another icon (representing a program that is able to handle the dropped object). For example, the icon of an image file can be dragged using the mouse and dropped upon the icon of an image viewer application, which then opens and displays the image file. [Wik13c]

Drag & drop is a common concept that is utilised in many, but not all, applications. It is a fast and easy-to-learn technique. Its usability may be degraded by lacking visibility of what can be dragged or dropped upon and lack of immediate feedback. [Wik13c, WCO95]

It bears similarities to the clipboard and the cut/copy and paste operations, since drag & drop is also used to transfer data from a source to a destination location, using a temporary buffer to store the data while it is being dragged.

Pick-and-Drop

This work is based on the Pick-and-Drop technique, which itself is an enhancement of drag & drop. In 1997 Rekimoto described drag & drop and its problems as follows:

> *"With the traditional drag-and-drop technique, a user first "grabs" an object by pressing a mouse button on it, then "drags" it towards a desired position on the screen with the mouse button depressed, and "drops" it on that location by releasing the button. This technique is highly suitable for a mouse and widely used in today's graphical applications.*
>
> *However, simply applying the drag-and-drop to pen user interfaces presents a problem. It is rather difficult to drag an object with a pen while keep* [sic] *the pen tip contacted on the display surface. It is often the case that a user accidentally drops an object during the drag operation, especially when dragging over a large display surface."* [Rek97]

As Rekimoto notes, this technique does not translate well to pen input. In fact, it does not translate at all to a number of other input modalities like keyboard-only input, voice

input and body gestures. Traditional drag & drop is clearly optimised for use with computer mouses, where the object can be dragged by comfortably holding down the mouse button while moving.

Although it is possible to drag & drop with touch input reasonably well – disregarding the problem of accidentally losing contact to the touch surface while moving larger distances (as has been noted by Rekimoto for pen input as well) – it becomes a problem when dragging objects across device borders, for example from one touch screen to another. [SSRG12, SSR13]

Pick-and-Drop proposes to solve this by replacing the drag action with the *pick* action: The user picks up an object by tapping it with the pen tip and then lifting the pen from the screen. The pen then "holds" the object, although in fact it is stored on a server and the pen is just used for user identification. This is similar to a clipboard and a cut & paste operation. The user can then move around freely and as soon as the tip of the pen comes close enough to another screen, a shadow of the picked up object appears at the cursor to indicate that it still holds the object. When the user then taps on the screen, the object is dropped.

This metaphor feels very natural, as it is similar to a person picking up an object, holding it while moving around in the physical world and then finally dropping it someplace else at the desired location. Therefore, the Pick-and-Drop concept was chosen as a basis for this interaction concept.

Gestures

Gesture recognition is a wide field that is commonly used to denote recognition of body and hand gestures, which is also the case in this work. Typically, cameras are used to capture body motion and touch screens are used to capture finger movements. Mathematical algorithms then try to recognise predetermined gestures which in turn trigger some kind of reaction on the computational device. [PKH08]

Gestures are a way to enrich current user interfaces to allow humans a more natural way of interaction with computers. Furthermore, they allow the development of new hardware and interaction concepts, such as touch screens, stereo cameras, wired gloves or even holographic displays that require no computer monitor, keyboard or mouse. [PKH08, WLD08, SDD12]

There are still many challenges, like the need for robust recognition algorithms, the handling of false positives when gestures are triggered by accident, the limitation to one source of input signal (e.g. body or hand) [SDD12] and usability issues like the visibility of gestures. When the set of available gestures is not visible, the user is left to guess or required to learn the gestures in order to use them. This is not intuitive and affects the ease of use of gesture input concepts, which may pose a serious problem for end users.

2.1.4 Ubiquitous Computing

> *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."*
> — Mark Weiser [Wei95]

This opening sentence of Mark Weiser from his article *"The Computer for the 21st Century"* describes the idea of ubiquitous computing very well. Ubiquitous computing describes a computing concept that allows human-computer interaction to occur everywhere, anytime, using any device without the user realising that he is using a computational device. For example, it allows the user to move from room to room without interrupting her interactive session with a computational device [BB00].

Ubiquitous computing relies on a variety of devices in different sizes that communicate with each other using different kinds of network adapters for short and long range communication and a support for different kinds of input and output concepts that fit the context. But the technological aspects are not the point of ubiquitous computing. The point is that these technologies seamlessly integrate themselves into the user's environment and seemingly disappear, leaving only the user and the goal he wants to achieve. [Wei95]

Additionally, *"users should not be required to go to a special place (i.e. the desktop) to interact with the computer. Nor should they be required to wear special devices or markers to have the computer know where they are."* [BB00] According to Brumitt et al., the computer needs to be aware of the physical world, based on a geometric model, to make appropriate decisions when selecting the device(s) to use. Otherwise, when computers do not possess a comprehension of the physical world, the proliferation of smart devices will lead to an increase of the complexity in the user's experience instead of simplifying it. [BB00]

The same is true for the input and output modalities utilised. Since mobile devices are changing locations frequently [BB00], adaptions to their surroundings are necessary to avoid the utilisation of improper modalities like speech input in a loud environment. This also requires certain knowledge of the physical world by the device and would benefit by knowledge about the user to make smart choices.

Another point touched upon by Brumitt et al. is the extensibility of the system, being able to grow automatically when new devices and capabilities are added [BB00]. This is similar to distributed computing, where the structure is not known in advance and may change during runtime.

2.2 Multimodal User Interfaces

A multimodal user interface (MUI) is defined by Oviatt as follows [Ovi03]:

> *"Multimodal interfaces process two or more combined user input modes (such as speech, pen, touch, manual gesture, gaze, and head and body movements) in a coordinated manner with multimedia system output. They are a new class of interfaces that [. . .] incorporate one or more recognition-based technologies (e.g. speech, pen, vision)."*

It is noteworthy that Oviatt calls the combination of output modalities "multimedia system output". In this work it will be called *multimodal output* for reasons of consistency.

A MUI represents a paradigm shift away from deterministic unimodal GUIs to probabilistic systems that started back in 1980 with Bolt's "Put That There" demonstration system. Bolt's system allowed users to create and move objects in a 2D GUI by combining speech in parallel with touchpad input [Bol80]. There has been great progress in the last decade in terms of hardware and software concerning MUIs with the proliferation of mobile devices like smartphones or tablets.

2.2.1 Architectural Overview

Compared to regular GUIs, there are quite some differences between them and MUIs, which is summarised in table 2.1.

GUI	MUI
Single input stream	Multiple input streams
Atomic, deterministic	Continuous, probabilistic
Sequential processing	Parallel processing
Centralised architectures	Distributed & time-sensitive architectures

Table 2.1: The differences betweeen GUIs and MUIs in juxtaposition. [DLO09]

In MUIs, different modalities can be used and combined for user input and output. This is a contrast to the deterministic input of GUIs, like mouse position or characters typed with a keyboard, where there is a single input that can be determined with absolute confidence. The multiple input streams in a MUI, on the other hand, have to be first interpreted by probabilistic recognisers and thus their results contain a certain degree of uncertainty. [Ovi03, DLO09]

Figure 2.1 illustrates the general architecture of a multimodal system. The orange-colored components introduced in the illustration are explained in the following sections.

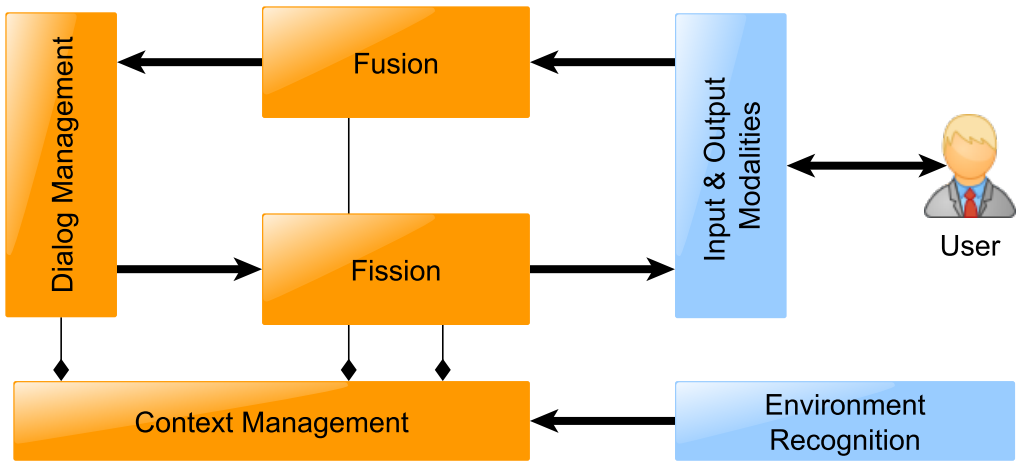

Figure 2.1: The general architecture of a typical multimodal system according to [HSW+13, DLO09].

Fusion

The individual results of the input recognisers have to be combined in a sensible way and mapped to a semantic representation, which is done by a component called the *fusion engine*. There are some challenges when mapping the inputs, since the input events are not always clearly temporarily delimited and there are other cases where temporal coincidence does not mean that these inputs are related to one another. [Ovi03, DLO09]

Moreover, uncertainty has to be taken into consideration when mapping the input results. However, the ability to provide different inputs for the same intended semantic representation adds robustness through redundancy, e.g. pointing at a selection and saying its name.

Dialog Management

When the inputs have been consolidated and semantically mapped by the fusion engine, they are typically sent to a component called the *dialog manager*. A dialog manager is responsible for the state and flow of a user interaction. Its purpose is to control the dialog between the user and computer so the user can achieve his goal.

When the dialog manager receives the semantic representations by the fusion, it identifies the current dialog state, executes the corresponding transitions based on the input and returns a message to the fission. [DLO09, HSW+13]

Fission

The fission is responsible to communicate the message received from the dialog manager to the user. It reasons about the most adequate modality or combination of modalities to communicate the message. This transforms the message from a modality-independent into a modality-specific representation. To do this, the fission customarily incorporates additional information about the environment, user and context. [DLO09, HSW12]

Context Management

A *context manager* is in charge of tracking things like the location, context and user profile for the fusion, dialog management and fission. This enables these three components to make smart, adequate decisions based upon this knowledge. [DLO09, HSW+13]

Advantages

MUIs aim to allow users a more “human” way of interacting with a computational system, since humans naturally support their communication with bodily gestures and facial expressions when talking to one another. They support more flexible, efficient and expressive means in HCI and are expected to be easier to learn and use [Ovi03].

But, as has been shown in studies, task completion time is only sped up by 10% and should not be considered the main advantage of MUIs [DLO09]. However, MUIs offer better reliability than other HCI techniques and bring more robustness to an interface [DLO09, Seb09]. It has been shown that users made 36% fewer errors when using MUIs compared to unimodal systems [DLO09]. MUIs also allow a more graceful recovery from errors through the ability to switch the input modality when an erroneous input was made. This are all user-centered reasons for error avoidance. There are system-centered reasons for superior error handling as well: *“A well-designed multimodal architecture with two semantically rich input modes can support* mututal disambiguation *of input signals.”* [Ovi03]

Finally, MUIs offer a choice of modality and allow the consideration of user preferences. For these reasons, most users prefer multimodal interaction over unimodal interaction [DLO09]. Moreover, choice of modality becomes important when certain modalities are ill suited or inappropriate for a certain task or environment, like keyboard input for drawing something, speech input in a library or for people with special limitations.

2.2.2 State of the Art in Multimodal Interaction Technology

Current research on multimodal user interfaces, like adaptable UIs, migratory UIs and model-based UI generation will be briefly introduced in this section. This is the current state of the art and represents topics that are related to this work.

Adaptable User Interfaces

Adaptive and customisable UIs use modelling and reasoning about the problem domain, the task and the user in order to better support the user in achieving his goal. *“The goal of such systems is to adapt their interface to a specific user, give feedback about the user’s knowledge, and predict the user’s future behavior such as answers, goals, preferences, and*

actions." [Seb09] Adaptive HCI promises to support more sophisticated and natural input and output that enables users to perform potentially complex tasks faster, with greater accuracy and to improve user satisfaction.

It is believed that this has a great potential for improving the effectiveness of HCI, therefore playing a major role in multimodal HCI. Several studies provide empirical support that the user performance can be increased when the UI matches the user's skill level. But there is also a concern that adaptive interfaces violate standard usability principles, with evidence suggesting that static interface designs sometimes promote superior performance than adaptive UIs. [Seb09]

Migratory User Interfaces

Migratory user interfaces allow users to switch devices while retaining their interaction session. This aims to make an application follow a user from one device to another, allowing the user to continue to work on the task he has been working on at the initial device, from the point where he left off with any input data being preserved. [BP04]

It may also include the adaption of the UI to fit different devices using an appropriate modality and presentation. These modalities can be used alternatively, complementarily or in another complex style of interaction [BP05]. The main reason for this is the diversity in devices and features, like different screen sizes or interaction capabilities, which must be considered for migratory UIs to be practical [BP04].

The time it takes for an application to migrate with its current state is crucial for usability, as has been noted by Bandelloni et al. Although the processing load to migrate an application with its state is heavy, users have been found to be disappointed when the migration was not perceived as instantaneous. User disorientation may be another problem, which could occur when the migrated UI does not resemble the initial UI. [BP04]

Model-based User Interface Generation

A model-based approach exploits one or more models as a basis to dynamically generate a UI at runtime. This allows the system to build multi-targeting UIs, which are better adapted to their target than premade UIs would be.

The models are declaratively described using device-independent markup languages and mechanisms of abstraction. They are transformed into concrete widgets by generators as defined by the target platform. [Vel09] Usually, specified generation rules and constraints are considered during the UI generation, like HCI design principles [Pet10]. This is beneficial for ubiquitous computing systems, since there can be no “one size that fits all” and context-awareness is considered one of the key points [CWC05]. Using a model-based approach can reduce the effort and complexity in creating UIs for many target platforms.

According to Vellis, model-based UIs are typically built for single-user, non-collaborative usage and have been dealing with rather simplistic problems far away from producing professional, fully functional applications [Vel09]. Because it is not considered mature yet, model-based UI generation is currently not used by many interface developers and has not been widely adopted by the industry [SGP00]. There is research on providing better notations and tools for the model-based approach [Vel09, CWC05], improving the application of HCI principles [Pet10] and supporting advanced use cases like collaborative UIs [Vel09] or distributed UIs [VC04].

In spite of everything, model-based UI generation is considered an important, necessary step for software development in an ubiquituous computing era [VC04, CWC05, Vel09]. Adaptive and migratory UIs are good examples for UIs that benefit from a model-based approach where the UI is generated at runtime.

Differentiation

To contrast this work from similar and related work, this interaction concept focuses on the possibilities and handling of multimodal information in a modality-independent manner across device borders.

There is existing work on **cross-device interaction**. Schmidt et al. [SSRG12] propose different interaction styles specifically tailored to mobile devices and touch screens or tables called “surfaces”. It introduces a variation of Pick-and-Drop that uses a phone as replacement for the pen to allow data exchange between phones and surfaces. Another technique is called “PhoneCopy&Paste”. It uses the phone as a personal clipboard to manipulate the surface’s content. Seifert et al. [SSR13] describe an approach to extend the screen space of a mobile device by utilising external screens as an UI extension. This allows data exchange and sharing using drag & drop across the devices.

In contrast to this work, the devices are limited to mobile devices and surfaces in both works. The focus lies on touch input; multimodal input capabilities are not considered. Also, the drop context is not taken into account to adapt the content's modality, it is exchanged as-is. The focus of [SSRG12] is to allow a seamless interaction between mobile devices and surfaces spanning device borders, similar to this work. [SSR13], however, aims to alleviate the inconvenience inherent to small screen spaces on mobile devices. This is of no concern for this work.

Other related work includes **Pick-and-Drop** by Rekimoto [Rek97], on which the interaction concept of this work will be based. Pick-and-Drop by Rekimoto is a direct manipulation technique that is an enhancement of drag & drop. It uses two actions, namely *pick* and *drop*, to exchange data between devices. After picking data up, the user is free to execute other actions or move around and switch devices, unlike in traditional drag & drop where the drag needs to be maintained – which is impossible in most multi-device environments. Rekimoto utilised pens that are identified by IDs to allow multiple users. Pick-and-Drop also uses a server called the *PenManager* in the background, that keeps track of which pen has currently picked up what data. The data itself is also stored on the server; the pens themself do not store any data. This work adopts most of the concepts developed by Rekimoto, but enhances them by removing the dependency on pens. It also enhances the pick capabilities by advanced features like the ability to pick multiple items. Pick-and-Drop also does support neither multimodal input nor multimodal output capabilities.

Miller and Myers already proposed a **synchronised clipboard**, whose contents are transferred over the network between computers [MM99]. The synchronised clipboard is hidden from the user and unobtrusively works like a typical cut/copy and paste operation, with the slight difference that the clipboard's content will be automatically synchronised across all configured computers. It uses a peer-to-peer architecture to achieve this. This work will use and enhance this approach by adding support for multimodal input and output and advanced features like the ability to collect multiple items to cut/copy from different sources. It will also feature a GUI and other feedback capabilities, making the hidden clipboard and its contents visible to the user.

Another work that is based on **copy and paste behaviour** of users was done by Stolee et al. with the goal of *"Revealing the Copy and Paste Habits of End Users"* [SER09]. Stolee et al. identified common habitual patterns and expressed some suggestions which were considered in the design of this interaction concept.

Finally, some of the proposed gestures that will be introduced later are inspired by the **boomerang technique** [KI07], which allows suspending drag & drop interactions to perform other actions while still maintaining the drag. It also offers advanced features like storing multiple objects and a gesture to receive the stored objects for dropping.

2.2.3 Existing System

The existing prototypical system realises a MUI, which generates its UI at runtime utilising a model-based approach. This allows the system to adapt the UI to the user, device and context, among other things.

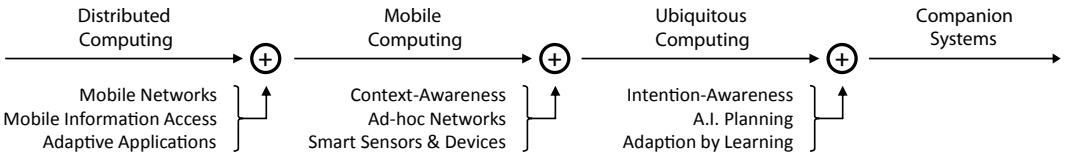

Figure 2.2: The evolution chain towards *Companion Systems* taken from [HSW12].

The system was realised as part of a Companion System. A Companion System is an advancement of ubiquitous computing, as illustrated in figure 2.2. Companion systems aim to adapt their behaviour to the user's preferences, needs, capabilities, emotional state, and situation. They possess intention-awareness, artificial intelligence planning and adaption by learning. [HSW12]

An overview of the relevant aspects concerning this work is given in the following section.

Architectural Overview

The overall architecture of the existing system is shown in figure 2.3. As can be seen, the fusion, dialog management and fission are located at the core of the system. Environmental information is collected by various sensors and stored in the *knowledge base*.

The *application core* stands for the application itself which provides the functionality, like a calendar application. *Task planning* represents the AI planning component that is responsible for providing a solution to given problems in the form of different tasks the user has to execute in order to accomplish his goal. These abstract tasks are handed to the *dialog management*, which in turn decomposes them into concrete and individual dialog

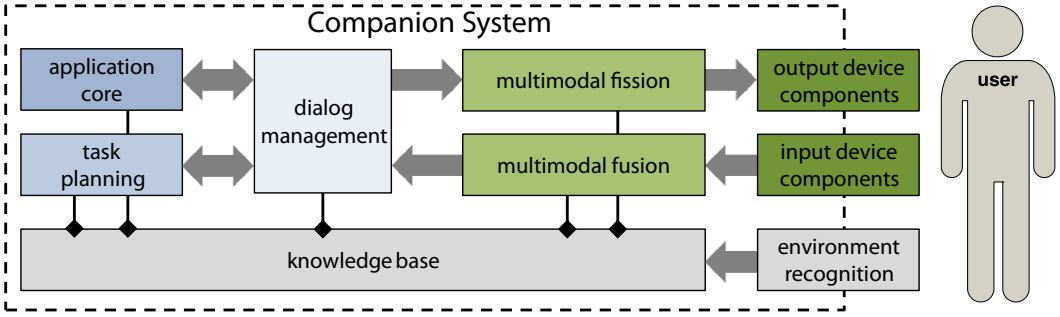

Figure 2.3: The overall architecture of the existing system. [HSW+13]

goals. Every communicable dialog fragment is then passed as a dialog act to the *fission* module. The fission assigns available modalities in a context aware manner and utilises output device components in order to communicate the dialog to the user. The user may then perform inputs that are captured by different input devices. The possibly multimodal inputs are interpreted by the *fusion* module, which maps it to a semantic representation in an adaptive process. This interpreted input is then handed back to the dialog management for evaluation. [HSW+13]

SEMAINE API

The existing system uses a message-oriented middleware (MOM) called the *SEMAINE API* to enable communication between components and devices in a distributed environment.

The SEMAINE API uses *ActiveMQ*, an open-source MOM which implements the Java Message Service (JMS) specification. The SEMAINE API uses a publish-subscribe model based on *topics* for communication: *"components can publish (send) messages to a topic or subscribe to the topic to receive messages sent to that topic. In other words, it is a flexible mechanism for n-to-m communication. To establish a communication between two components, it is sufficient for them to use the same topic name."* [SEM]

SEMAINE provides a system monitor, shown in figure 2.4, that displays a graph of all components as circular nodes and the topics as rectangles, with edges between the components and topics to indicate paths and direction of communication. The SEMAINE system monitor can furthermore show exchanged messages for any given topic by clicking the topic rectangle which opens a new window that displays all messages being sent on the topic.

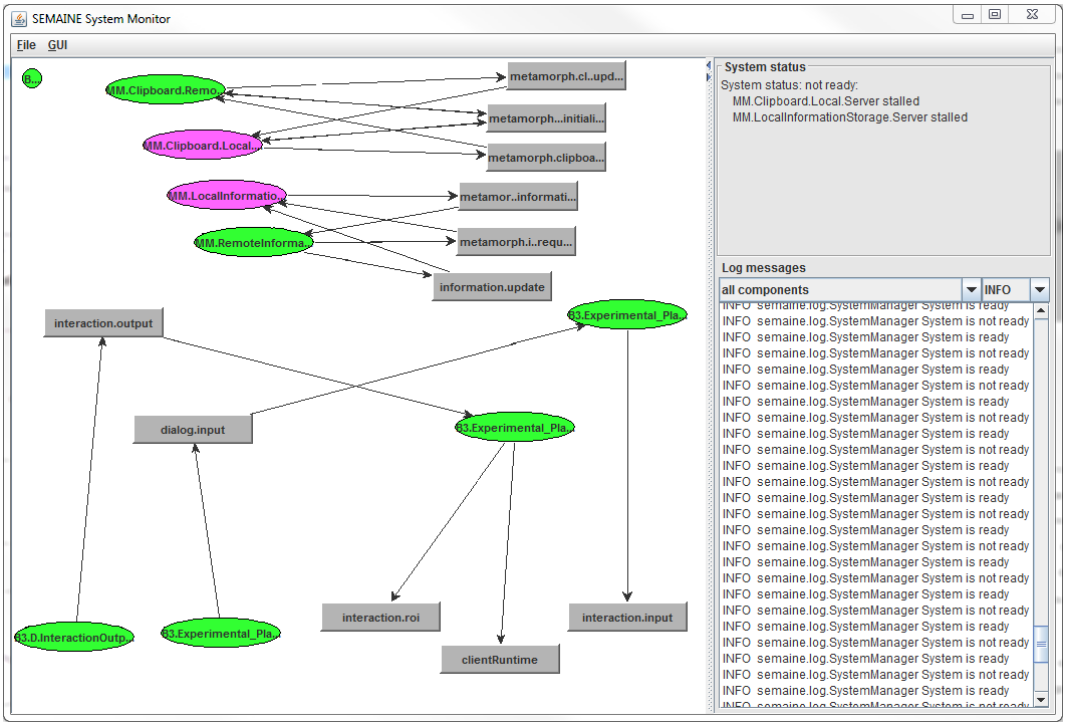

Figure 2.4: The SEMAINE System Monitor providing an overview of all topics and nodes in the system.

3 Interaction Concept

Multimodal user interfaces (MUIs) fit the human way of interacting better than traditional graphical user interfaces (GUIs) since they allow an interaction style that feels more natural to the users, are seen as being more robust in terms of error prevention and shall allow easier and faster error correction [Ovi03, DLO09, Seb09]. To unfold their full potential, MUIs need to support flexible and dynamic content as well as multiple interaction and display modalities that adapt to the current context of use [ZNK10]. The proposed interaction concept aims to gain further insight into MUIs using a drag-&-drop-like interaction style to exchange information in a multi-device environment with a model-based approach.

The theoretical concept, the prototypical implementation as proof of concept, and its evaluation are the main contribution of this work and will be elaborated in the following chapter. But before describing the interaction concept itself, the scope of the concept will be detailed to set the framework for what shall be achieved and what is beyond the scope of this work. Furthermore, to demonstrate its practical applicability, a scenario with some use cases is explained before delving into the interaction concept itself in detail.

After the concept has been described, a first evaluation that has been done using a paper prototype will be detailed. The evaluation was done in the early stages – right after the first interaction concept draft was ready – to gain first insights, uncover hidden problems early in the design process and to elicit oversights and forgotten necessities in the first draft. This is elaborated on in section 3.4.

3.1 Scope

This section sets the scope for the interaction concept by defining objectives, deriving requirements and detailing the limitations.

3.1.1 Objectives

The overall objective of this work is to study a modality-independent handling and presentation of information and its exchange across devices using the Pick-and-Drop concept with model-based MUIs. Such an interaction concept has several interesting points to consider, which will be explained in the following paragraphs.

First, the information has to be stored in a modality-independent way with mappings to modality-specific representations that supports an exchange across device borders. To enable picking up the information from one of its different concrete representations and dropping it somewhere else in another concrete representation, the information has to be enriched with meta-information – to identify and define its available concrete representations. When picked up, there must be some kind of transition from its current concrete representation to its abstract representation as shown in figure 3.1. Furthermore, when dropping the information, there needs to be a transition from the abstract representation to one of its concrete representations as predetermined by the drop context. This process will be further detailed in section 3.3.1.

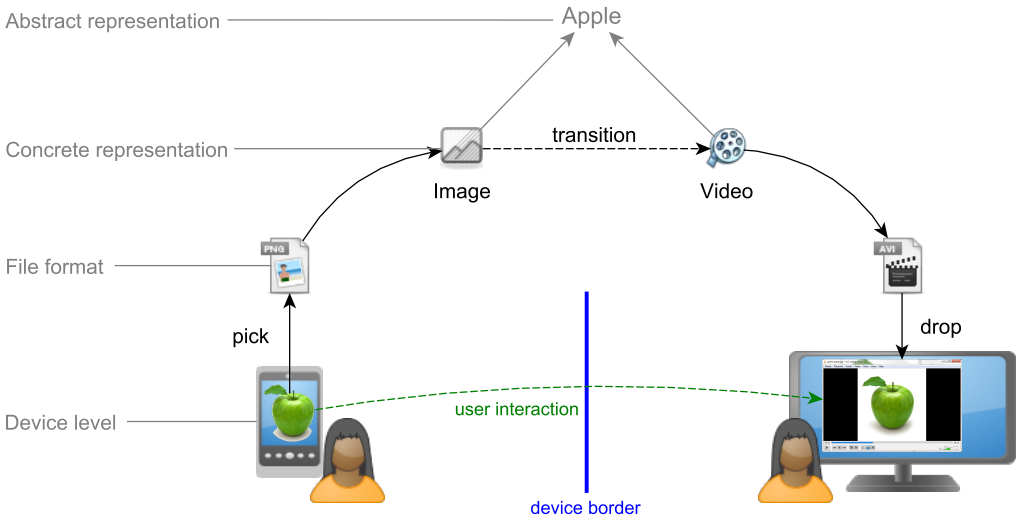

Figure 3.1: This illustration shows how the information transitions from one concrete representation (png image) to another (avi video). The represented information – an apple – stays the same. The Pick-and-Drop interaction also crosses the device borders from a smartphone to a touch screen.

Second, the interaction concept needs to be realised with modality-independence in mind. This means there needs to be a definition of what actions are offered by the interaction concept, what states and transitions these actions entail and how these actions can be triggered using different input modalities – or a combination of modalities. This will be based on the prevalent input concepts and the assessment of their strengths and weaknesses as is done in section 3.3.6 on page 48. Additionally, every feedback has to be defined for several output modalities as well. One goal is to enable this in an extensible and flexible way by decoupling the triggers from the actual actions they trigger by introducing an interaction concept with two levels: An abstract, modality-independent level and a concrete, modality-dependent level.

Third, the picked information can be dropped onto different UI components on the concrete level – as long as those components support one of the media types offered by the picked abstract information. According to [WCO95] drop site feedback is under utilised, but is something that can aid users and make the system more predictable. Also, possible drop targets are not inherently obvious by just looking at the information as an abstract concept. For these reasons, it is necessary to identify possible drop targets and indicate them to the user utilising the most appropriate modality available on the target device. This means the drop-target indicator itself has to be presented in different modalities.

Last but not least, transitioning between different kinds of information representations is not always naturally possible, depending on the source and target representation. There are inherent ‘gaps’ in transitioning from some representations to others that have to be identified and considered in the interaction concept.

3.1.2 Requirements

The objectives detailed in section 3.1.1 lead to the following list of requirements:

- Pick-and-Drop-like interaction with multimodal information items
  - Description & storage of multimodal information items
  - Ability to transition information items from concrete representation to abstract representation and vice versa
  - Context-sensitive drop to determine representation to use
  - Identification & handling of problem cases with multimodal Pick-and-Drop

- Exchange of information across devices
  - Allow other actions to be taken between *pick* and *drop*
  - Clipboard-like concept
    - Synchronised in distributed environment
    - Ability to store multiple information items in clipboard
    - Ability to select/deselect an item information in the clipboard
    - Ability to examine current items and the selected item at any point
  - Spontaneous change of modalities to support different devices
- Modality-independent interaction concept
  - Abstract interaction concept that models the possible actions, states and transitions decoupled from specific modalities (abstract level)
  - Modality-dependent mappings for all supported modalities that detail what inputs trigger which action (concrete level)
  - Feedback (abstract and concrete level)
  - Drop target indicators (abstract and concrete level)

3.1.3 Limitations

There are also some limits that apply to this concept, either due to technological or conceptual constraints.

Since the information content does not affect the interaction concept at all, all information content and its corresponding meta-information is pre-authored and not generated or automatically detected, as this is not the focus of this work. Although there are known approaches for low and high-level adaption and procedural generation of content, this will be deliberately not utilised in this work.

Furthermore, the context information will be pre-authored as well, meaning there will be no usage of a dynamic context-sensitive determination of output modality, like an adaptive agent or (machine) learning algorithm. Another aspect is that user skills or preferences won't be taken into consideration by this interaction concept.

The input modalities will be limited to prevalent concepts. No concepts requiring complicated fusion, like the combination of speech with pointing gestures in a “Put That There” type of manner [Bol80] will be realised in the proof-of-concept prototype and only touched upon in the theoretical interaction concept.

3.2 Scenario

This scenario aims to support applications based on the Ottawa Charter for health promotion, first developed by the World Health Organization (WHO) in 1986. The use cases are aimed to illustrate possible applications of the interaction concept in a context of promoting healthy nutrition and assisting physical training. They are presented using the following template:

Situation: <Description of the initial situation where the user wants to achieve a goal>

Proposed solution: <Description of the steps taken to achieve the goal>

Abstract information: <Notation of the concepts or things with which the user interacts>
Concrete information: <List of concrete representations that are used in the use case, in the form of: `name-of-representation` (`type-of-representation`)>
Input modalities: <List of utilised input modalities>
Output modalities: <List of transitions the information goes through, in the form of: `source-representation` → `target-representation`>

Use Case I

Situation: The user is reading about the equipment necessary to do Kendo on his computer. He reads about something called a *“shinai”* but cannot imagine what this is.

Proposed solution: The user selects the text, picks it up and drops it onto an image viewer to have a look at it.

Abstract information: Shinai[1]
Concrete information: Name (atomic text), photograph (atomic image)
Input modalities: Mouse, keyboard
Output modalities: Text → image

[1] A Shinai is a weapon used for practice and competition in kendo representing a japanese sword.

Use Case II

Situation: The user wants to compile a nutrition schedule on her tablet without much writing effort or the need to research the nutrition details.

Proposed solution: The user selects the food she wants in his schedule by picking up images of the food from a grid view and dropping it into her word processor on the computer to print it out afterwards.

Abstract information: Food_n [2]
Concrete information: Photograph (atomic image), name and nutrition details (atomic text)
Input modalities: Touch, mouse, keyboard
Output modalities: Image $\rightarrow$ text

Use Case III

Situation: The user is browsing a video-on-demand website on his desktop computer and finds a movie he wants to watch later on while working out on his home trainer.

Proposed solution: The user picks up the text title or the cover image of the movie und drops it later onto the video player of the home trainer by using its touch screen.

Abstract information: Movie_n
Concrete information: Title (atomic text), cover (atomic image), movie (hierarchical video file)
Input modalities: Mouse, keyboard, touch
Output modalities: Text $\rightarrow$ video OR image $\rightarrow$ video

Use Case IV

Situation: The user wants to try out a new yoga technique. He finds instructions on the yoga technique with his smartphone on the internet, but has problems following the written instructions.

[2] *n* is an index used to denote a specific item out of a collection, like an apple out of a fruits collection.

Proposed solution: The user chooses to pick up the instructions on his smartphone and drop them on the TV with the XBox Kinect in his workout room to watch them.

Abstract information: Yoga technique
Concrete information: Website (hierarchical text), video (atomic video)
Input modalities: Touch, voice, gestures
Output modalities: Text $\rightarrow$ video

Use Case V

Situation: The user is reading a fascinating eBook on her tablet from which it is hard to tear yourself away, but she wants to go jog outside.

Proposed solution: The user picks up the eBook on the tablet and drops it via a voice command onto her MP3 player, where it will be continued as an audio book on the page the user left off.

Abstract information: Book
Concrete information: eBook (hierarchical text), audio book (hierarchical audio)
Input modalities: Touch, voice
Output modalities: Text $\rightarrow$ audio OR text $\rightarrow$ text-to-speech

Use Case VI

Situation: The user wants to plan a hiking trip.

Proposed solution: The user picks up the start and destination from a map on a tablet and drops them into the corresponding text fields of a hiking planner on his PC using speech commands. He also picks up gear he wants to take with him by using a collection of pictures showing the relevant gear. The labels for the gear text fields adapt themselves to fit their content. The 'start' and 'destination' labels, however, stay the same.

Abstract information: Travel destination$_{(start,\ dest)}$, hiking gear$_n$
Concrete information: Locations on a map (hierarchical picture, atomic text), hiking gear (atomic picture, atomic text)
Input modalities: Touch, voice
Output modalities: Image $\rightarrow$ text

3.3 Metamorph Interaction Concept

This interaction concept is called *Metamorph*, since it seemingly **morphs** the information item's representation to fit the context of the drop target location using **meta**-data that defines what representations to use in which context. This concept will be described in detail in the following sections.

3.3.1 Basic Concept

The Metamorph interaction concept adopts the Pick-and-Drop approach [Rek97] to enable a cross-device exchange of information, but removes the fixation on pen input. Metamorph is designed to be usable and adaptable to a number of different input and output modalities, thus extending the Pick-and-Drop technique to a general, multimodal and model-based interaction concept.

This will be achieved by decoupling the basic actions from the actual, modality-dependent triggers. *Pick* and *drop* as basic actions will be kept, as well as the concept of storing the picked up objects in a clipboard-like manner. The clipboard will be queryable by using the supported modalities and offer advanced features (as described in section 3.3.2). The means to trigger the pick and drop actions will be fitted to the respective input modalities (see section 3.3.4) as is the user interface feedback to the respective output modalities (see section 3.3.5).

Abstract Interaction Concept

The abstract interaction concept describes the Metamorph interaction concept at its most basic, modality-independent level. It represents the core of the interaction concept. To achieve this, the most basic elements and terms that are used need to be defined first. Table 3.1 defines the basic elements.

For convenience reasons, the information item will be abbreviated to just **'item'** and the Metamorph clipboard will be just called **'clipboard'** from here on.

The clipboard shall offer up to six slots in which items can be stored. This number is based on the work of Miller in *"The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information"* [Mil56]. It may be possible to have a bigger,

smaller or dynamic number of slots for the clipboard, but such preferences and their impact on usability are not the focus of this work. This could be the subject of a future work.

Element	Description
`Cursor`	The term cursor is used to denote the location designator for triggered actions. It can be either a graphical cursor visible on the screen, like a mouse cursor, or an invisible cursor, like when using a finger to point at something.
`Pickable`	Marks an abstract or concrete component as valid target for `pick` actions.
`Droppable`	Marks an abstract or concrete component as valid target for a `drop` action.
`Information item`	An information item is anything that can be picked up and dropped using the Metamorph interaction concept. It can range from a simple text to an image to a video, to name a few examples. An information item is pickable.
`Slot`	A slot represents a storage unit for a single information item. A slot is droppable.
`Metamorph clipboard`	The Metamorph clipboard is the globally available storage for the picked up information items, offering slots in which information items can be stored. Although the storage itself is invisible, it is made visible through audio feedback and GUI components that are described later.

Table 3.1: Overview of the basic elements and terms that are used to define the Metamorph interaction concept.

This means the clipboard allows storing more than a single item at a time, opposed to traditional clipboards that can only store a single item like the Windows clipboard or the clipboards of most Linux distributions. Exchanging information across devices may necessitate moving larger distances between devices, depending on the device set-up and environment. To minimise tedious movement between devices to transfer several information items originating from devices at different locations, an activity that is rather common according to [SER09], the decision to support multiple slots was made.

Using those elements and terms, the Metamorph interaction concept's basic actions can be defined as described in table 3.2. The data required to perform the corresponding actions is also provided in the description represented as unordered n-tuples.

Concerning the required data's meaning, the `DeviceID` is an identifier of the device on which the action was triggered, the `ComponentID` designates the device's UI component (a device can have multiple UI instances running simultaneously, e.g. a GUI component, a text-to-speech (TTS) component, etc.) and the `ObjectID` identifies the concrete UI object (e.g. a text box). This 3-tuple is necessary to identify a concrete UI instance in a distributed system.

Action	Description
`Pick`	Store the information item picked up from ObjectID in the clipboard slot with SlotID, the next slot that is free or the slot that was least recently used. **Required data:** `<DeviceID, ComponentID, ObjectID, InformationID, [SlotID]>`
`Drop`	Drop the dragged information item into the designated UI object with ObjectID, initiating a transfer of the information to a droppable UI object. **Required data:** `<DeviceID, ComponentID, ObjectID, InformationID>`
`Select`	Mark the information item stored in the clipboard slot identified by SlotID as selected. **Required data:** `<SlotID>`
`Deselect`	Unmark the information item stored in the clipboard identified by SlotID as selected. **Required data:** `<SlotID>`
`ClearSlot`	Clears the slot identified by SlotID by removing the information item it contains from the clipboard. If no SlotID is given, the item in the selected slot will be removed by default. **Required data:** `<SlotID>`
`ClearUiComponent`	Clear the source UI component with ObjectID, making the drop a cut instead of copy. **Required data:** `<DeviceID, ComponentID, ObjectID>`

Table 3.2: Overview of the basic actions defining the Metamorph interaction concept.

The `InformationID` identifies the affected information item. The `Cursor` is a positional information from which the ComponentID and ObjectID can be derived. To pick an object up, most often the cursor will be used to identify the item. Consequentially, the general purpose of the cursor is the identification of UI objects to interact with. This information may be gained otherwise, for example by stating the name of the item using voice input to identify it. Because of this, the cursor is not listed on those occassions. The `SlotID` identifies the slot of the clipboard.

Figure 3.2 illustrates the procedure when picking up items. An item can be picked up either using a cursor to identify the item to pick up or by directly identifying the item. To pick an item up using the cursor, it must be placed over it. The user can then start to drag the item around and drop it someplace he desires – either over a clipboard slot to pick it up or at another location that supports to drop items on it. Only cursor-based techniques support the concept of dragging an item, therefore dragging is considered a modality-specific feature and not listed in the basic actions of the Metamorph interaction concept to maintain modality-independence. It is also possible to directly pick the item up to store it in a slot. When an item has been picked up, it is by default automatically selected.

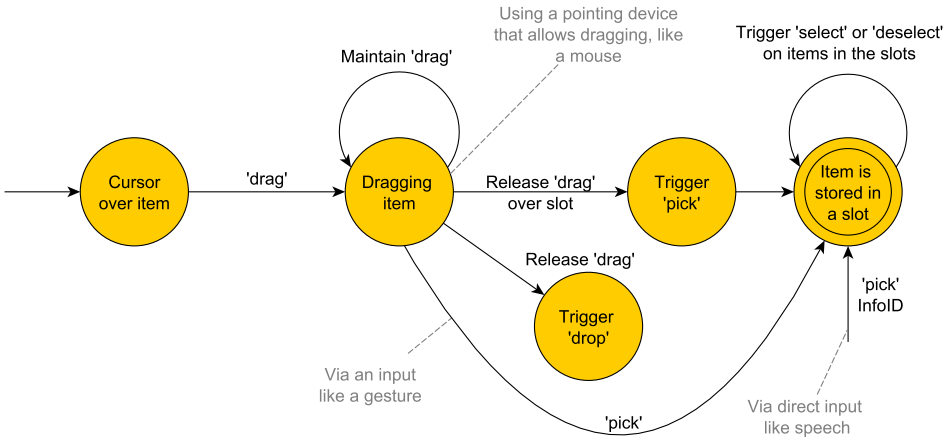

Figure 3.2: A state diagram that illustrates the possible actions when picking up an item.

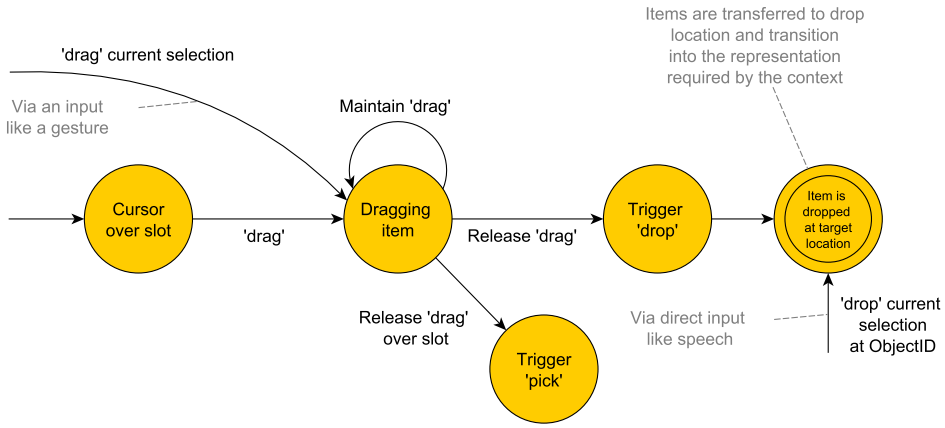

Figure 3.3: A state diagram that illustrates the possible actions causing the drop of an item.

Dropping an item is illustrated in figure 3.3. To drop an item, the user can either select it in the clipboard and then drop it or avoid the clipboard by initiating a drop directly from its original location to its destined location. The drop action automatically transitions the item into the concrete representation specified by the drop target. If an item does not possess the required representation, it cannot be dropped onto the drop target, which will be indicated by some kind of modality-specific feedback.

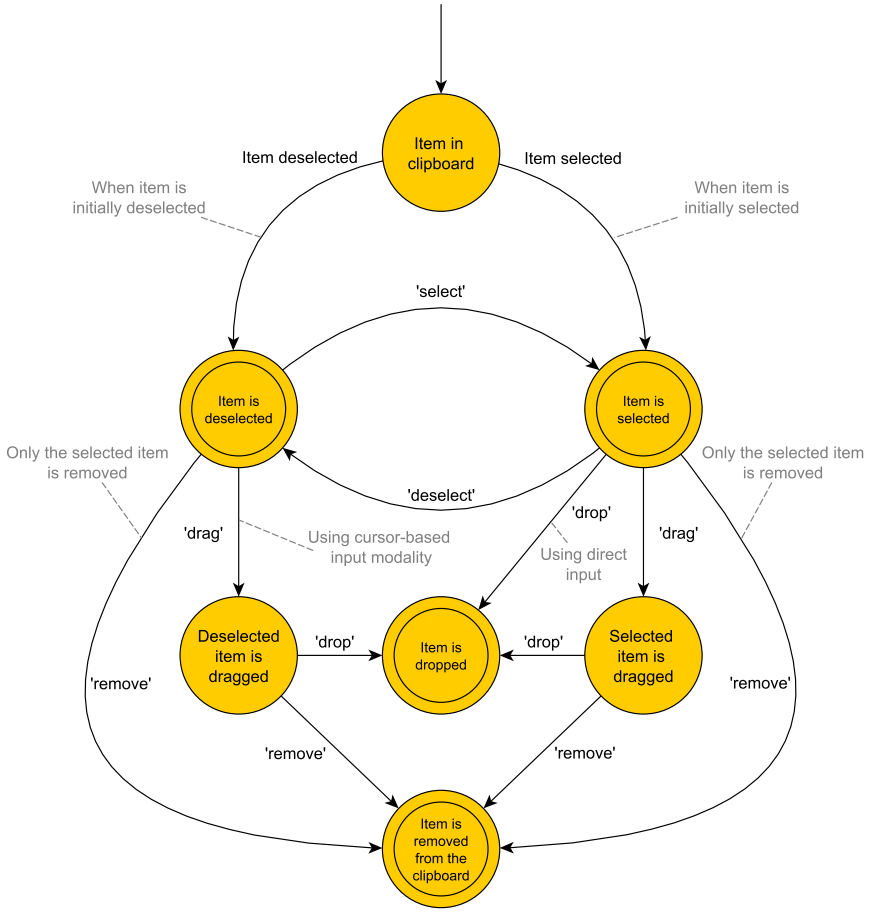

Figure 3.4: This state diagram illustrates the interactions and transitions for item selection.

Figure 3.4 illustrates the transitions and states of changing the selection of items in the clipboard and how the selection affects the drop action. Items can be selected and deselected however the user desires by triggering the `select` or `deselect` actions. Selected items will be indicated depending on the modality.

By default, when no item is specifically designated, the drop action uses the currently selected item from the clipboard. Items can also be removed from the clipboard by triggering the `remove` action.

Representations of Information

Information can be communicated in different ways, depending on the 'thing' or concept they represent. For a multimodal user interface, being able to choose between different representations to represent the same 'thing' allows the MUI to select the best fit in a multi-device environment to enhance the user experience.

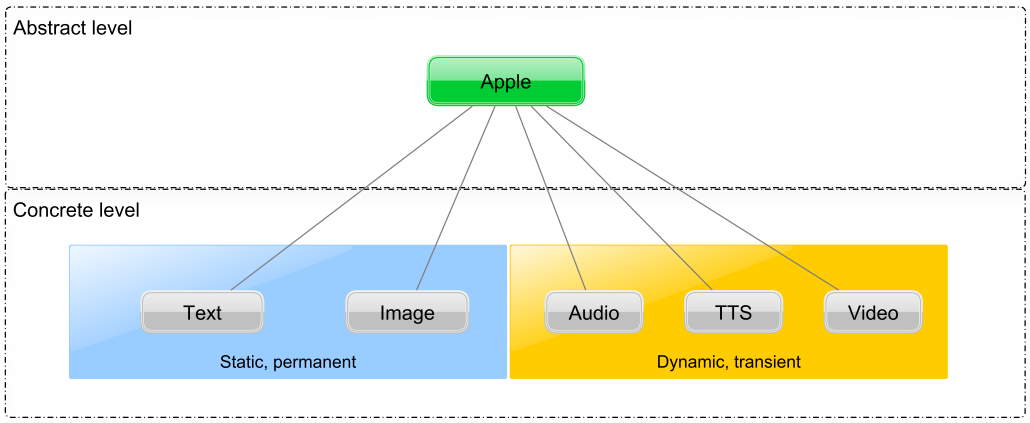

Figure 3.5: A diagram that illustrates the two levels to describe information on a computational device and in the minds of people.

To enable this, information has to be somehow classified, structured and described by meta-information. Figure 3.5 depicts how the conceptual idea of an apple can be represented on two different levels: The abstract level and the concrete level. All representations stand for the same 'thing', although being expressed by different media representations and being stored in different file formats on computational devices.

The **abstract level** resides within people's minds and describes the apple as a general concept: An apple is a fruit which grows on apple trees, it can be eaten, is nutritious and is even considered to be beneficial for a person's health. Although there exist different kinds of apples, with different properties like taste, colour and shape, people assign all of these different representations to the concept of an apple. Furthermore, an apple may be

represented by its name, like *Granny Smith*, by its conceptual name (“apple"), by photos, drawings, people pronouncing the words “Granny Smith" or “apple", a video of an apple and many more. All of those things are associated with the concept of an apple. This level is inherently abstract, therefore the representation of information at this level, in style of the Cameleon Reference Framework [VLM+04], is called the *abstract representation*.

The **concrete level** resides both within people’s minds and within the bounds of computational devices. A medium (singular of media) describes the storage and transmission channels used to communicate information or data, meaning written text, vocalised words, pictures, drawings, moving images or even gestures [Wik13d]. People are able to distinguish and identify different media easily. Also, media is heavily used by computational devices to store information or data in different ways. There are five types of media considered by this interaction concept: Text, image, text-to-speech (TTS), audio and video. This level limits the communication channel by concretising the representation; therefore it is called the *concrete representation*.

Data Transfer Mode

There are two possible data transfer modes: *Cut* and *copy*. Depending on the context and the user input, either one is used. According to Stolee et al. in [SER09], which explored how end users utilise a default operating system clipboard, 45% of the interactions captured in their study were copy operations, 52% were pastes and only 3% were cuts. For this reason, the default action will be copy on drop.

A **cut** moves the data, meaning it is removed from its source location and added to the target UI component. When it is removed, the corresponding representation of the source information is changed accordingly, meaning it will be blank after the move (e.g. when cutting a text from a text field, the text representation of the source information will be changed to be blank). The target’s information representation will be changed to the moved data. Cut, on the concrete level, is possible only when a flag called `IsEditable` is set to `true` for the abstract source object and when it is requested by user input. The cut option is always available for slots.

A **copy** creates a new identical instance of the data at the target location, meaning after the copying is finished, the data will be present in the source as well as in the target. The information content of the target will be changed to the copied data, whilst the source con-

tent will remain unchanged. Copy is executed by default or when the source component's `IsEditable` flag is set to `false` since this prohibits a cut operation.

Nominators

The label that is typically placed just before a UI component to describe its content will be called a *nominator* in this work. A nominator can be a GUI label or a spoken hint for text-to-speech output.

A nominator will be overridden on a drop action when the `OverrideNominator` flag at the drop target component is set to `true`. This may be necessary for cases where GUI components need to adjust their semantic meaning to their content, like when dropping a vegetable into a text field nominated as "Meat:" (due to the last item dropped there) when collecting a list of food to compose a meal. In other cases, where the nominator gives a new semantic meaning to the UI component, like when assigning an address as travel destination, the nominator "Travel Destination:" should not be changed to something like "Address:" in this example.

A cut will only remove the nominator when the `OverrideNominator` flag is set to true, as the meaning of the nominator is related to the moved information data.

3.3.2 Abstract Components

The Metamorph interaction concept uses a number of components to achieve its goals. These components follow the principles of the model-view-controller (MVC) design pattern by separating the application into three types of components: Components handling the application data, control and presentation. This is illustrated in figure 3.6. These components are described in the section at hand.

Metamorph Clipboard

The Metamorph clipboard is the core component on the abstract layer that remains in the background and connects everything. It represents the model and offers six slots for information items to be dropped on and picked up from, acting as a centralised storage that

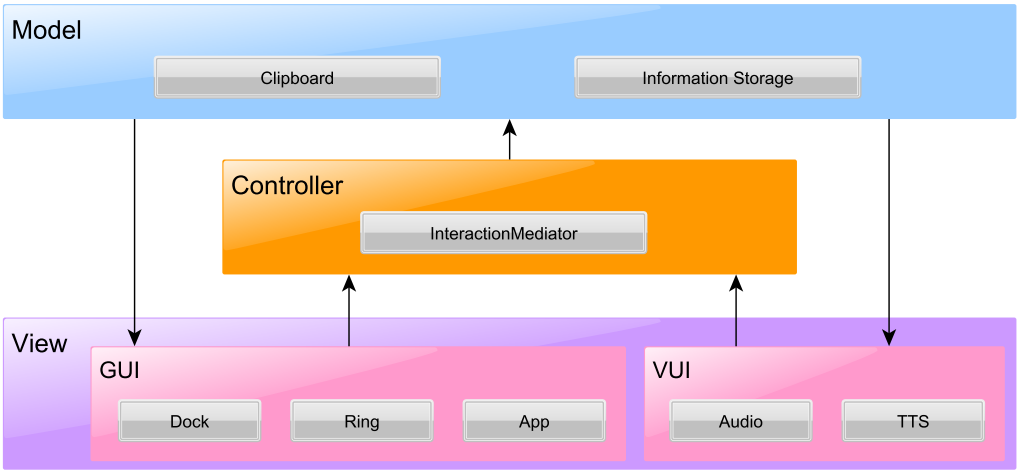

Figure 3.6: The model-view-controller structure followed by the interaction concept.

is synchronised across all devices. The Metamorph clipboard replaces the traditional clipboard in the Metamorph interaction concept. The other components rely on this central storage component to display the current contents in the slots and the state of the items in the slots, as well as their relevant meta-information like available representations on the media level.

The clipboard itself is hidden from the user, but made visible through other components depending on the device and interaction style.

Information Storage

The information that is handled and exchanged by the Metamorph interaction concept needs to be stored somewhere in a centralised manner. It has to be accessible from anywhere in the distributed environment and needs to be kept in sync on all devices and UI components. Updates to the information need to be applied everywhere. The `InformationStorage` is a model that manages, stores and synchronises the information.

Interaction Mediator

The interaction mediator models the basic actions of the abstract interaction concept (see 3.3.1 on page 30). It represents the controller that receives requests to trigger the basic

actions of the interaction concept and acts as an encapsulating layer between the diverse input modalities and the executing components. It handles requests, as defined by the MVC, by delegating them to the responsible components that can execute these actions.

Views

One of the main advantages of the MVC pattern is the ability to freely interchange and add new ‘views’. Although this term is typically used to denote a visual presentation, the MVC does not limit the views to GUI representations. This is true for the Metamorph interaction concept as well, which allows the clipboard to be represented in various different ways, including the auditive channel. Some proposed possibilities are shown in section 3.3.3.

3.3.3 Concrete Components

In this section, concrete example components are described to give a feeling of what views are imaginable to be realised for the interaction concept.

Metamorph Dock

The Metamorph dock is a bar at the top or bottom of the screen. It represents one possible way of visualising the six slots offered by the clipboard that is proposed in this work. In addition to the six slots, there are three buttons to the right side of the slots stacked upon each other that allow to hide the dock, configure the dock settings and remove the selected or dropped upon item (from top to bottom). See figure 3.7 for a draft of the dock, illustrating several different states.

The dock is always visible to avoid confusion on the user side. It always shows the clipboard’s content and what the user can do with the items in the dock, indicated by small general media icons below the item’s name. This follows the advice on visibility given by Donald Norman in [Nor02]. The media icons tell the user what representations are available on every slot. It may be thinkable to add a user preference that allows to auto-hide the bar for proficient users that prefer to have more screen space, but user configurability is not a key aspect of this work. Items that are dropped onto a slot are automatically ’selected’ according to the basic actions described in section 3.3.1 on page 30.

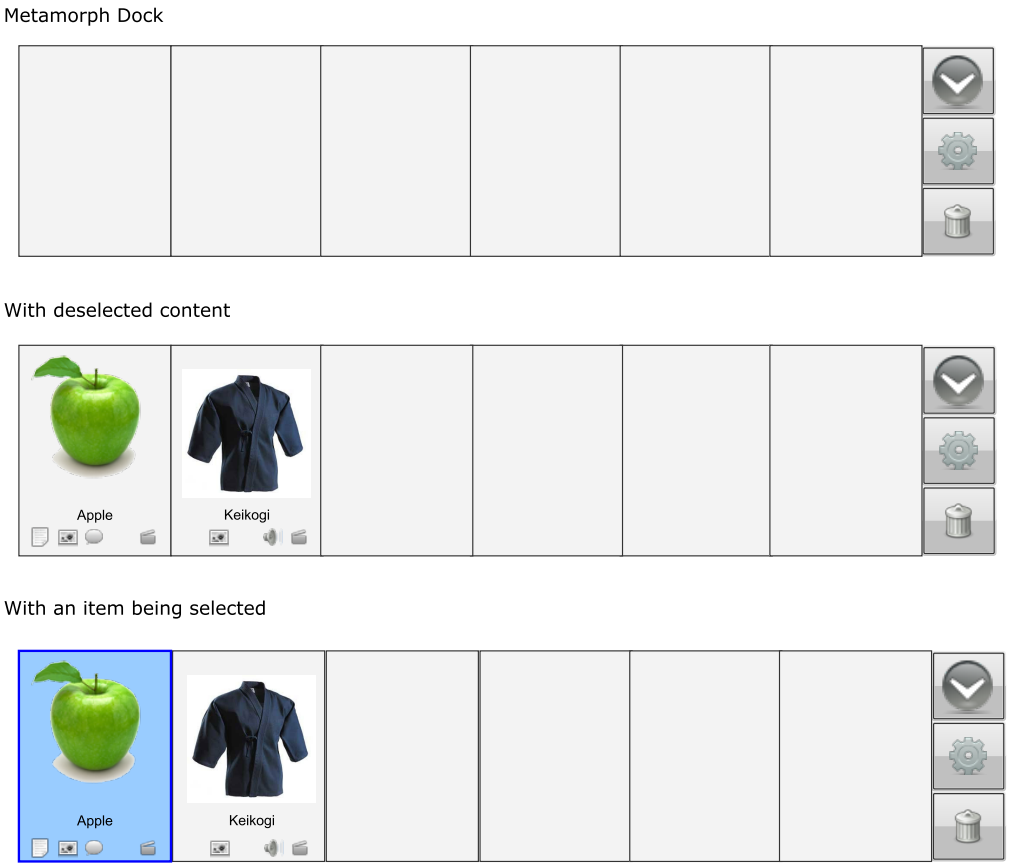

Figure 3.7: A concept draft of the Metamorph dock, depicting the overall basic look of the dock and with two information items in the slots, the first being selected. The icons below the information items indicate what representations are available.

The dock is intended to be used in computer environments that offer a little bit more screen space, like desktop computers, laptops, touch tables or wall-mounted touch screens.

Metamorph Ring

The paper prototype in section 3.4 revealed a problem with the dock and its static position at the top or bottom of the screen for touch tables. On horizontal screens like a touch table, it is ambiguous what 'top' or 'bottom' means, as this is dependent on which side of the table the user is standing.

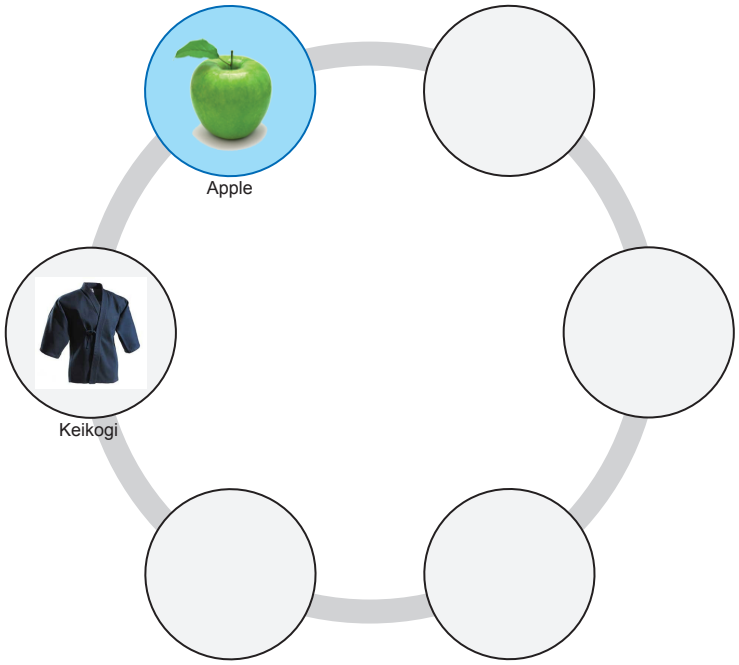

Figure 3.8: A concept draft of the Metamorph ring.

Furthermore, when multiple users want to work on a touch table in different areas, a fixed position might be inadequate. To alleviate these problems, the Metamorph ring is proposed as a possible solution.

The ring offers six slots for items like the dock, but in a ring shape centered on the cursors location at the time the ring is summoned. It appears when executing some specific actions and disappears when it loses cursor focus. The contents in the slots are in synchronisation with the clipboard, meaning the items and their state in the slots will always be the same.

The ring allows local and efficient access to the items in the slots. One problem that remains is the determination of what is ‘up’ or ‘down’ for a user standing before a touch table, to show the icons and text at the correct angle. One possible solution, when the ring is summoned by a gesture, could be to detect the user’s position by determining the angle of the gesture. Another issue with the ring is to find suitable places for the media type icons on the ring. A possible solution might be to place them around the slot circles.

Metamorph App

Figure 3.9: A concept draft of the Metamorph app.

Components like the dock or ring are aimed to be added to a screen that displays other information alongside these components. Therefore, they are aimed at systems that offer moderate to large screen space. On small screens, they claim too much screen space, leaving no room for additional information. Furthermore, they are not optimised for small screens. The Metamorph app is a proposed solution for small screens.

The app offers six slots in a fullscreen grid of 2x3 slots and is kept in synchronisation with the clipboard. There is a menu from which the app settings can be accessed and a selected item can be removed from the slots. The following components are displayed in the slots: An icon of the item, the items name and, below the name, several small general icons. Those small general icons indicate what representations are available for the given item.

The app is intended for devices with small screen space, like smartphones, tablets or other mobile devices.

Voice User Interface (VUI)

When no displays are available, a voice user interface (VUI) becomes necessary. Utilising text-to-speech, the current content of a clipboard containing two items, one being selected, could be expressed like this: *"There are currently two items in the clipboard. Slot 1: Apple. Supported representations: Text, picture, video. Slot 1 is selected. Slot 2: Keikogi. Supported representations: Picture, audio, video."*

Actions to query individual slots, the current selection specifically or a list of possible drop targets can be realised as well utilising TTS.

Multimodal User Interfaces

A combination of GUI and VUI, where both complement each other or provide redundant information are possible as well. A complementing UI could utilise the GUI for the basic information that is needed most of the time, whilst the VUI provides additional in-depth information on top of the GUI when requested. A redundant combination allows using the GUI to gain a fast overview, while using the VUI when looking at the screen is not possible or comfortable.

Adding acoustic sounds to actions or events might also increase the usability, like a sound played by devices that support the currently picked item to highlight them as possible drop targets. This also helps the user to locate the physical device.

3.3.4 Modality Mappings

The three most important actions constituting the essential core of the Metamorph interaction concept are considered here: `Pick`, `drop` and `selection`. These actions will be elaborated on in the following sections.

Also, the mappings pictured in this section will be limited to the following input concepts: Mouse, keyboard, touch screen and voice. These input concepts are considered to represent the most important input concepts that are covered here due to their popularity and prevalence.

For a description of the remaining actions and other input concepts see the appendix, where the complete version of the tables are printed.

Pick

Table 3.3 describes the steps that need to be performed to trigger the `pick` action.

The description of the voice commands needs some explanation: Everything within the quotation marks is intended to be spoken. Square brackets mark optional words or phrases, like `[up]`. The text enclosed between double angle quotes are placeholders for objects of the UI, like the current «`selection`» in the clipboard.

Input method	Steps to trigger 'pick'
Mouse (GUI)	1. Click on the desired item with left mouse button 2. Hold the click and start dragging the item i. The Metamorph ring appears 3. Release the click over a slot
Mouse (gesture)	1. Click on the desired item with left mouse button 2. Hold the click and start dragging the item 3. Perform a swipe gesture to the top i. The item will appear in a slot in the Metamorph dock
Keyboard	1. Select the desired item 2. Press Control (ctrl) + x for cut or ctrl + c for copy i. The item will appear in a slot in the Metamorph dock
Touch (GUI)	1. Long touch on the desired item i. The item will be visually 'picked up' i. The Metamorph ring appears 2. Release the touch over a slot
Touch (gesture)	1. Select the desired item i. Item will be visually selected 2. Perform a swipe gesture to the top i. The item will appear in a slot in the Metamorph dock
Voice	1. "Pick «`selection`» [up]", "Take «`selection`»", "Collect «`selection`»" i. Some auditive or visual feedback confirms the pick-up

Table 3.3: A detailed description of the actions needed to perform the pick action using a set of input concepts.

To capitalise on the mental model people already have of picking something up, all gestures use a swipe gesture to the top – to emulate the actual physical act of picking an object up.

This is called a ’natural mapping’ by Norman and, according to him, “leads to immediate understanding" and helps to remember the actions necessary to trigger the pick action. [Nor02]

Drop

The steps necessary to trigger the drop action are described in table 3.4.

For the voice command, «ObjectID» is referring to the drop target UI component's identifier. This could be something like its nominator, for example.

Input method	Steps to trigger 'drop'
Mouse (GUI)	1. Click with the left mouse button on the desired item 2. Hold the click and start dragging the item 3. Release the click over the desired location to drop the item
Mouse (gesture)	1. Click and hold the left mouse button 2. Perform a swipe gesture to the bottom i. The item in the selected slot appears at the cursor 3. Release the click over the desired target location to drop the item i. Perform shaking gesture to cancel the drag
Keyboard	1. Press ctrl + v i. The item in the selected slot will be dropped at the cursor's current location
Touch (GUI)	1. Long touch on the desired item i. The item will be visually 'picked up' 2. Release the touch over the desired target location to drop the item
Touch (gesture)	1. Perform a swipe gesture to the bottom i. The item in the selected slot appears at the finger's position 2. Release the touch over the desired target location to drop the item i. Perform shaking gesture to cancel the drag
Voice	1. "Drop [selection] at «ObjectID»", "Put [selection] into «ObjectID»"

Table 3.4: A detailed description of the actions needed to perform the drop action using a set of input concepts.

To be consistent with the pick action, the mental model people have of dropping an object is utilised for the drop action. This is achieved by relying on a swipe gesture to the bottom and emulates the actual physical act of depositing an object on a table or dropping an object on the ground.

Selection

The procedure to select or deselect an item that is stored in a clipboard slot is detailed in table 3.5 for the given set of input concepts.

Input method	Steps to trigger 'select' and 'deselect'
Mouse (GUI)	1. Click with the left mouse button on an item i. If the item is deselected it will be selected ii. If the item is selected it will be deselected *Alternative* 1. Rightclick and select 'Metamorph ring' from the context menu i. The Metamorph ring appears 2. Click to select or deselect as described before
Mouse (gesture)	1. Perform a circle gesture i. The Metamorph ring appears 2. Click to select or deselect as described under 'Mouse (GUI)'
Keyboard	1. Press Alternative (alt) + Tab until the dock is focussed 2. Use the arrow keys to focus the desired item 3. Press spacebar or enter i. If the item is deselected it will be selected ii. If the item is selected it will be deselected *Alternative* 1. Press ctrl + r i. The Metamorph ring appears 2. Select or deselect using the keyboard as described before
Touch (GUI)	1. Touch on a deselected item i. The item becomes selected 2. Touch on a selected item i. The item becomes deselected
Touch (gesture)	1. Perform a circle gesture i. The Metamorph ring appears 2. Click to select or deselect as described under 'Touch (GUI)'
Voice	"What is in the clipboard?", "What have I picked up?", "Show [me the] contents of the clipboard" *Current Selection* "What [item] is [currently] selected?", "Show|Tell me the selection" *Change selection* "Select|deselect slot «ID»", "Select|deselect «position, like first, second, etc.» item", "Select|deselect item in slot «ID»"

Table 3.5: A detailed description of the actions needed to perform the select or deselect action using a set of input concepts.

3.3.5 Feedback

Feedback is an important aspect of any user interface. Without feedback, users cannot know if their actions were registered or what the outcome of their actions has been – either if it was successful or ended up in failure. Lacking this information, the users cannot gain an understanding of the system, therefore hindering them in achieving their goals. [Nor02]

Output modality	Feedback	
Visual	**Pick:**	• Item appears at the dock in a slot
	Drag:	• Icon of the picked up item is shown at the cursor
	Drop:	• Item appears at the drop location • A plus icon appears at the cursor when the drop will be a copy
	Item selected:	• Frame and background of the selected slot become hightlighted with a colour
	Item deselected:	• Frame and background of the deselected slots become as usual
	Clipboard contents:	• Metamorph dock (global) • Metamorph ring (local)
	Selected item:	• Frame and background of the selected slot is highlighted with colour
	Representations supported by the items:	• Small general icons below the item's icon that represent text, image, text-to-speech, audio and video representations • Detailed tooltip information on mouse over slot
	Valid drop targets:	• Coloured frame around possible drop targets
Auditive	**Pick:**	• A specific sound with increasing tone height to imply a upward movement
	Drag:	—
	Drop:	• A specific sound with decreasing tone height to imply a downward movement
	Item selected:	• A selection sound
	Item deselected:	• A deselection sound
	Clipboard contents:	• Enumeration of all items by TTS
	Selected item:	• Description of the selected item by TTS
	Representations supported by the items:	• Enumeration of the modalities for a specific given item
	Valid drop targets:	• Enumeration of all possible drop targets' names by TTS

Table 3.6: An overview of feedback for various actions, subdivided into the visual and auditive channels.

The various kinds of feedback and how the feedback is given depending on the output modality is shown in table 3.6.

As can be seen, the options to provide complex information with auditive feedback are rather restricted to mostly text-to-speech output or audio cues (sometimes also called *auditory icons*).

3.3.6 Transitions

There are two kinds of transitions that take place when using the Metamorph interaction concept: The transitions of input modalities and the transitions between information representations.

The transition between input modalities happens, for example, in minor cases when a user switches from using the mouse to using the keyboard on the same device. Or, in a more significant case, when a user transitions from using touch input on a tablet to using voice and body gesture input in front of a wall-mounted screen. This includes switching devices as well. Such details severely affect the interaction concept and how the user interacts with the devices, simply because using a keyboard or mouse is completely different from using voice input or body gestures. Not every interaction concept translates well – or at all – to other input concepts. For example shortcuts, which are common when using a keyboard, are very difficult to translate and use with body gestures.

When the representation of the information item transitions, the media type of the information the user is working with needs to be switched. Examples include picking up the textual representation of an information and dropping it into a video player to play its video representation or picking up an image and dropping it into the audio player to hear an auditive description of what the image is showing. The following sections illuminate what restrictions, possibilities and problems have to be considered for these transitions to work.

Input Concepts

Input can be roughly divided into two kinds of input: Input providing **spatial information** and input providing **semantic information**, like text or commands. This distinction is rather rough, as spatial information may also be used to convey semantic meaning, for example a drawn picture is an accumulation of strokes in a structured way or when using gestures to trigger commands. But typically, these are the two crucial inputs needed to operate a device efficiently.

This distinction is required to examine the transitions between input concepts: Each input modality has its own characteristics, offering different strenghts of expressive power for conveying spatial or semantic information. Some devices may be more apt to convey spatial or semantic information than others are.

In the case of transitions between input modalities, the expressive power has to be compared and the question needs to be asked if the other input modality allows the user to input and express the same information as the original one. Table 3.7 depicts a matrix that contrasts every input modality with every other to answer this question.

	Mouse	Keyboard	Touch	Voice	Body gestures
Mouse	Equivalence	Complementary	Specialisation	Complementary	Redundancy
Keyboard	–	Equivalence	Specialisation	Redundancy	Complementary
Touch	–	–	Equivalence	Complementary	Redundancy
Voice	–	–	–	Equivalence	Complementary
Body gestures	–	–	–	–	Equivalence

Table 3.7: Transitions between some input modalities to one another. The table should be read from left to right, rows to columns.

Four kinds of cooperative relationships, borrowed and adapted from the TYCOON framework [Mar99], have been assigned to describe how transitioning between one input modality to another affects the expressive power of the user.

These four cooperative relationships are defined as follows:

- **Equivalence:** Both input concepts are approximately equal good at the same kinds of input.
- **Specialisation:** Both input modalities can be used for the same inputs, but one is better suited to do so than the other and always to be preferred over the other.
- **Redundancy:** Both input modalities can be used for the same inputs, but they might have individual strengths and weaknesses differing from one another.
- **Complementary:** Both input modalities are strong at one kind of input and weak at another, they complement one another and thus are best used in combination.

As can be seen in table 3.7, an input concept is – naturally – equivalent to itself.

A transition from mouse to touch is a **specialisation**: Both input concepts are apt for providing spatial information, but a touch screen is better suited to input text than a mouse,

since multiple fingers can be used to input the letters on a virtual keyboard. Whereas with the mouse, only one letter can be moved to and clicked at a time – which is rather cumbersome. It may be argued that a mouse is more precise at pointing than a touch is, but this is beyond the scope of this work – both work reasonably well at pointing [MSB91]. A transition from keyboard to touch is also a specialisation since a keyboard can be emulated on a touch screen and touch is better at providing spatial information than a keyboard. Although the keyboard is better at inputting text, since it is physical and provides haptic feedback [FWW11], the touch input is overall more apt to provide spatial and semantic input. Additionally, haptic feedback may be achieved on touch screens using vibration on touch or other approaches.

Body gestures form a **redundant** relationship with the mouse: Both can be used for pointing and performing gestures, but the mouse is more precise at pointing and body gestures, as the name implies, work better for gestures to trigger commands. Also, they are rather weak at inputting semantic information like text. Keyboard and voice input form a redundant relationship as well: They are both equally effective at inputting text, with voice input being significantly faster and offering more ease of use [ZNK10]. Special characters might be easier to input using voice (for novice users not familiar with a keyboard or characters not found on a typical keyboard), but it may be overall more prone to errors than keyboard input. Finally, touch and body gestures are redundant as well with touch being the superior input concept. Semantic input is rather hard to input using body gestures, especially text [HDS12], whereas spatial information can be inputted with both reasonably well with each concept having their inherent strengths and weaknesses: Body gestures allows quicker pointing, especially over larger distances, whereas touch is more precise.

Input concepts that form **complementary** relationships are only apt for one kind of input, spatial or semantic, and thus need to be used in conjunction with another input concept that provides an apt method for the other kind of input. Mouse and keyboard are a well-known combination of input concepts that complement one another well, using the mouse as a precise pointing device and the keyboard to input semantic information. The mouse also has a complementary relationship with voice input, since voice input is severely lacking in expressing spatial information. Text and commands, however, can be naturally and efficiently inputted using voice – provided that the speech recognition works flawlessly [HDS12]. Although uncommon, keyboard and body gestures complement each other as well, with the keyboard being used for semantic input and body gestures mainly for spatial information. The same goes for touch and voice input, with touch being used to countervail the lack of

spatial expressive power of the voice input. Lastly, a more common approach is to combine voice input with body gestures, which is especially popular in today's science-fiction user interfaces and in research projects [HDS12].

Media

The media transitions depicted in table 3.8 have been identified by creating a matrix of the atomic media types identified in section 3.4.1. The media types can be split into two categories: Permanent media and transient media.

Permanent media is static and consists of media representations like text or images. Those types of media always look the same and present their information all at once in a persisted state that does not change over time.

Transient media is dynamic, like audio[3] or video. It only communicates a fraction of its information at a time. To get the complete information that the transient media contains, it needs to be played back over time. Also, when it has been played back, its representation has volatilised – it does not persist. Dynamic in this context, however, does not mean that the information itself changes or may be changed by the user – the played-back information stays the same, only its representation changes in a predetermined, reproducible pattern during playback.

		Target Representation			
		Text	Image	Audio	Video
Source Representation	Text	–	No complications	Permanent, static → transient, dynamic	Permanent, static → transient, dynamic
	Image	No complications	–	Permanent, static → transient, dynamic	Permanent, static → transient, dynamic
	Audio	Transient, dynamic → permanent, static	Transient, dynamic → permanent, static	–	No complications
	Video	Transient, dynamic → permanent, static	Transient, dynamic → permanent, static	No complications	–

Table 3.8: Transitions between source and target representations. Green indicates transitions without difficulties, orange indicates a transition from a permanent to a transient representation and yellow indicates a transient to a permanent transition, which have some inherent incompatibilities.

[3] Text-to-speech is considered a subtype of audio in this case.

A transition within the same category of media does not yield any complications. Picking a textual representation of an information and then dropping it as an image is unproblematic and the information persists, only in another representation.

The transition between transient types works also as exptected: When picking an audio information and dropping it into a video player, the video starts to play back just as the audio would have and when the video ends, the information output is over.

It gets problematic when a textual representation is dropped on the video player: The basic information is then “played-back” as a video and its representation suddenly becomes transient. When the video has been played-back, the information representation is “lost” as it has volatilised.

Identified Problems

The **handling of cut and paste operations** for transient media is inherently problematic. To provide an example, when a user picks up a music album on the computer and drops it onto his audio equipment, the question arises whether the information should be removed from the computer – to fulfill the contract of a cut and paste operation – or not. If it were to be removed, the information would be lost since the audio equipment does not have a storage for the music album, it just receives the stream from the computer.

Also, only atomic types have been considered up until now. Often though, **media is hierarchical** – text has headlines, subheadlines and paragraphs, audio has tracks and timestamps, audiobooks and videos are often structured into chapters, etc. When transitioning betweeen media representations, this additional information about the internal structure and current position of the user has to be stored and preserved during a *pick* and *drop* operation across devices and media types. Another problem is, that a mapping has to be found for a position, for example, in a video to the same position in a text: When the user just watched a specific chapter of a video, what position in the text corresponds to this position if there is a corresponding position at all?

There is a **gap during the transition between input modalities**: Some input modalities use a physical device and provide tactile feedback while others do not. A mouse, keyboard and touch screen are all physical devices that are used to receive input, whereas voice and body gestures input uses physical devices which are not directly involved in the user’s interaction, like a microphone or video camera. Switching from tactile input methods to

non-tactile may feel unnatural or strange. Since most dominant input methods up until now have been tactile, switching to non-tactile input modalities may also feel unfamiliar and may need some time to get accustomed to.

When making a **selection out of a pool of information items** by picking one up, the information data will be copied, as it makes no sense to remove it from the selected information as a consequence of selecting it. For example, picking the colour blue out of a collection of colours should not erase the text or image representing the colour from the information. A reference would be another possible solution, with changes being made to the information object transferring back to its original information and vice versa, thus the data would not be removed from the information upon selection.

But this would introduce a gap to the interaction concept, further complicating its usability as this behaviour does not fit into the behaviour of the interaction concept so far, with the UI components representing a given information, remaining independent of one another. Additionally, a reference introduces a relation between UI components, which is hard to make visible without cluttering the UI. But without a visualisation, a user cannot see that modifying a certain information item in one place changes the same information in all other places as well.

In most cases, a selection is not made to edit the information itself, but rather to compose multiple information items to create new information, in which case the selected information's data is copied into the new information fragment.

When working with information that can consist of only a subset of the available representations, the problem of how to **handle composing subsets of representations** arises. For example, when dropping an information item with a text and an audio representation on a UI object that holds an information that contains a text and picture representation, the text and audio representations are merged into the dropped information, while the picture representation is retained. The purpose of dropping information on UI objects is to compose new information, so it would not make any sense to remove the picture representation because the dropped information did not possess one.

Using the nominators of UI components to identify the UI objects when using ASR leads to an **inconsistency with dynamic nominators** that adapt themselves to the information they store (see section 3.3.1 on page 37). This behaviour may not be immediately apparent to the user.

3.4 Paper Prototype

To make the evaluation of the interaction concept possible in this early stage, whilst not even a single line of code had been written yet, the paper prototyping method was used [Ovi03]. Paper prototyping is a method that allows to evaluate a user interface in a user-centered design approach by using rough drawings and (potentially hand-drawn) sketches on paper to simulate a user interface prototype in an evaluation, as shown in figures 3.10 and 3.11. This is similar to a *Wizard of Oz experiment*, where the reactions of the computer are simulated by a human. However, unlike in a Wizard of Oz experiment, the human helper is visible to the test participants and helps to walk them through the tasks. [Pap13]

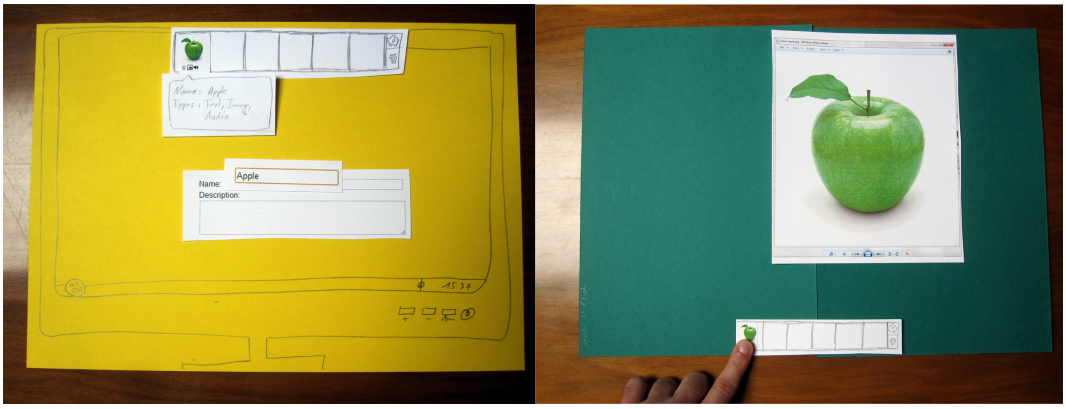

Figure 3.10: Photos of the paper prototype. On the left, a computer screen with the Metamorph dock and a form is shown. On the right, a touch table with the Metamorph dock and an image viewer is shown. The dock contains the single information item “apple" on both devices.

This approach offers several advantages [Med07]: It offers the ability to catch design flaws very early in the design process when it is easy and advantageous to fix them. Furthermore, no technical background is required by the evaluation participants. The usage of paper encourages experimentation, as paper can be complemented or modified by new ideas easily even on the spot with just pens and scissors. It allows fast and flexible iterations, which allows to change directions more easily since not much effort has to be invested into the paper prototype. Documenting the evaluation is easy, as it is possible to write notes directly onto the prototype. And lastly, the cost requirements to perform it are very low (both in budget and preparation).

The procedure as well as the summarised results are presented in the following sections.

Figure 3.11: Photos of the paper prototype for mobile devices. On the left, a tablet is shown, experimentally using the Metamorph dock instead of the app. On the right, a smartphone featuring the Metamorph app is shown.

3.4.1 Procedure

Before the actual paper prototype evaluation could be conducted, some preparations had to be done and some things to be considered. The following steps have been taken:

1. List input concepts to consider in the evaluation
2. List media types to consider in the evaluation
3. Create matrix of interesting or problematic media transitions (shown in section 3.3.6)
4. Devise questionnaire
5. Devise available information items to interact with
6. Devise user tasks
7. Conduct evaluation
8. Summarise and write down results

These individual steps will be detailed in the following subsections.

Input Concepts

After the initial draft of the interaction concept was written, the first step was to list all input concepts to include in the evaluation. The following input concepts – without any particular

order – have been considered: Mouse input utilising a GUI, mouse input utilising gestures, keyboard input, touch input on a large touch screen (like a touch table), touch input on a small touch screen (like a smartphone), speech input and body gestures.

This list covers a wide range of input concepts and possible devices that support those concepts, like desktop computers, notebooks (traditional, subnotebooks, tablet computers), tablets, touch tables, (wall-mounted) touch screens and smartphones to name the most common ones.

Atomic Media Types

Subsequently, the supported media types were listed as follows: Text (visual), image (visual), audio (auditive) and video (auditive and visual). These are oriented on the most basic types of media, with the exception of video – which is a multimedia type that combines images with audio, but this is not relevant for this evaluation.

Only atomic media types have been considered, meaning types without any hierarchical structures. Also, these media types are independent of any specific media formats like `WAV` or `MP3` for audio formats, as this does not affect the evaluation of the interaction concept.

Media Transition Matrix

See section 3.3.6 on page 51 for the identified matrix of possible and problematic media transitions.

Questionnaire

The questionnaire was done using the *Single Ease Question* (SEQ), where only a single question is asked concerning the ease of use after the given task was done. The participant can then rate the ease of use using a 7-point Likert scale ranging from “very difficult” to “very easy”.

The participants were asked to perform every single task using every input concept once and to rate it using the 7-point rating by answering the SEQ. Also, a field for additional notes was placed under the rating table to allow additional feedback.

Items

The following items were made available to use and interact with during the tasks of the paper prototype:

- `Apple`
  Supported media types: Text, image, audio
- `The Breakfast Club` (movie)
  Supported media types: Image, audio, video
- `Uni Ulm`
  Supported media types: Text, image, video

Tasks

The participants were asked to accomplish the following tasks:

1. Pick up `Apple` from one kind of type and then drop it on another type.
2. Pick up `The Breakfast Club` from the `Name` text field. Copy it to the image viewer and then to the video player.
3. Pick up `Apple`, `The Breakfast Club` and `Uni Ulm` from one kind of type. Select `Uni Ulm` and drop it on the video player.
4. Pick up `Uni Ulm` on the computer and drop it on the smartphone into the video player.
5. Pick up `Apple` on the smartphone and drop it on the video player on the touch table.

3.4.2 Results

A number of insights and feedback have been gained by the evaluation. First of all, it was clear that the Metamorph dock should always be visible to the user when it contains at least one item. This increases the clarity of the user experience and avoids confusing the user with a hidden multi-item clipboard. It has even been suggested to have it visible all the time instead of only when it contains items or when the user starts to drag an item.

Additionally, the observation was made that the speed of gestures will play a vital part concerning their usability. Triggering gestures by accident or having gestures that are difficult

to trigger even though the user intends to do so needs to be prevented in order to avoid user frustration.

Another observation was that having only a dock with item slots at the top of the screen may proof inappropriate for large screens, especially for large wall-mounted screens where reaching the top may be difficult or impossible. Also, the term “top” may be ambiguous for touch tables where users may stand at any corner they wish to. This led to the introduction of the Metamorph ring.

Furthermore, it was noted that voice-only identification of locations is very hard near to impossible. It is therefore preferable to combine voice input with other input modalities more fit for pointing to determine target items and locations. The evaluation was then altered to allow voice commands to be complemented by another pointing concept (like a mouse or pointing gestures using the body). In addition, some kind of feedback is especially important when making selections by using pointing gestures using the body.

It was also revealed that feedback on some actions, a possibility in the GUI to change the drop mode (cut or copy) and the possibility to change the selected items using keyboard, touch and gesture input was missing completely in the current concept. This was immediately amended after its discovery.

These results helped to improve the interaction concept by smoothing some rough edges and by pointing out some missing parts.

Identified Interaction Problems

Some inconsistencies, which can not be easily remedied, have also been revealed by the paper prototype.

On the one hand, using the control key (ctrl) to switch the data transfer mode (cut/copy) when dragging something – which is a convention on Windows systems – is not possible for cases where no keyboard is available, like a touch screen running a typical OS. Alternative inputs have to be utilised to allow this switch of modes when using other input modalities than a keyboard. Also, using the control key to indicate whether to cut or copy the information cannot be used in the keyboard-only case, as the established paste shortcut already uses the control key for triggering the paste operation itself.

On the other hand, it may be confusing to have a dock GUI at the top of the screen and additionally having a ring GUI at a local nearby position. But this is necessary since the picked-up items should always be visible, although having the ring be visible all the time might be bothersome. However, having a ring GUI is not fit for small screens like smartphones.

3.5 Interim Conclusion

This concludes the interaction concept. During the course of this chapter, the requirements and limitations have been defined, the abstract interaction concept with its basic elements and actions were introduced, the abstract and concrete components that constitute the interaction concept have been proposed and some inherent problems with offering a multimodal interaction concept have been elaborated.

The multitude of interaction possibilities offered by the Metamorph interaction concept come at the cost of simplicity, learnability and clarity. It is attempted to alleviate the complexity by utilising natural mappings, making the clipboard visible and by providing clear and helpful feedback to keep it usable. A prototypical implementation will help to evaluate the feasibility and usability of the interaction concept and its realisation.

Some possible problems have already been identified in section 3.3.6, which was one goal defined in the requirements from section 3.1.2 on age 25. For the prototypical implementation, problems like cut and paste with transient media or hierarchical media types will be ignored by limiting the prototype to copy-only. But since they pose an interesting question as to how users expect the system to behave in such cases, they will be examined in the expert evaluation following in chapter 5. Other problems, like the merging of information representations or dynamic nominators in conjunction with ASR, have been uncovered by the prototypical implementation and found their way back into the interaction concept. Because the merging of information is trivial, as it would minimise the use to simply override one information item with all representations of the other, it will not be further discussed. The handling of dynamic nominators in combination with ASR, on the other hand, poses an interesting problem and will be further examined in the expert evaluation.

4 Prototypical Implementation

A prototype is developed as a proof of concept, to gain solid first-hand experience and feedback on the proposed interaction concept from a user perspective and on the feasibility from a developer perspective. The prototype extends the existing system introduced in section 2.2.3 on page 19. Like the existing system, it was implemented using the C# programming language utilising the .NET Framework in version 4.0.

The prototype is mainly implemented in the output and input device components of the existing system, as depicted in figure 4.1. This is sufficient to proof the concept introduced in chapter 3 and short-circuits complex systems like the dialog management, the fission and the fusion in order to retain full control over all aspects in the UI. Nevertheless, the proper integration into the model-based multimodal system is implemented and described. Also, only the copy mode was implemented (see section 3.3.1 on page 36), since cut may not always be possible.

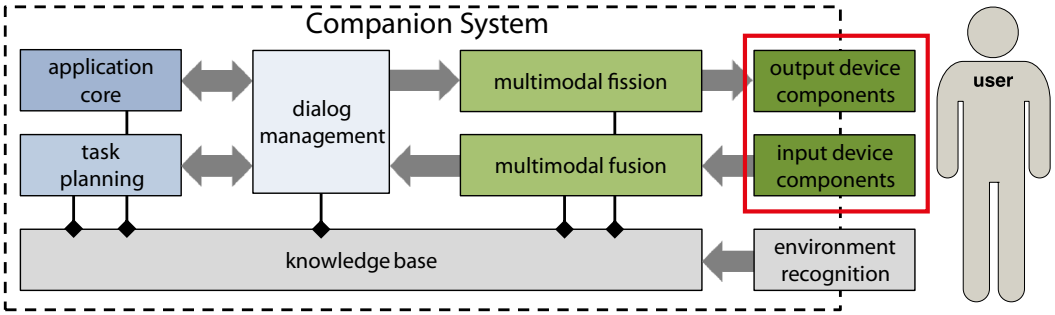

Figure 4.1: Architectural overview of the existing system. The prototype was mainly implemented and integrated in the output and input device components. [HSW+13]

This chapter starts by examining the existing system and its implementation in more detail. An understanding of some parts of the existing system is necessary insofar as the prototype utilises and extends it. The system design is introduced in the next section, followed by an elaboration on the implementation in the section thereafter. Lastly, a conclusion on the future prospects is given in the final section of this chapter.

4.1 Existing System

The existing system offers a model-based approach to user interface generation at runtime. It is a distributed system that uses the *SEMAINE API* [SEM], a message-oriented middleware (MOM), for communication. The system's interaction management (IM) allows the flexible and dynamic adaption of the user interface to the context of use at runtime. The IM also provides capabilities for automatic speech recognition (ASR), text-to-speech (TTS) output, an XML schema to model device capabilities, adapted GUI components based on the .NET Windows Presentation Foundation (WPF), an event-driven system for dialog handling and finally logging and monitoring capabilities for messages sent via the MOM, for input and output.

4.1.1 User Interface

The main assembly, that is concerned with the generation of the generic MUI, is called `ClientRuntime`. It accommodates the executable to start the client UI, the various UI components, the device model, the UI generation and code for generating dialog and interaction input messages. Figure 4.2 shows a diagram of its structure reduced to the relevant aspects for this work.

Dialog & Interaction Output

The **dialog output** represents the abstract UI and is described using XML files. It is a modality-independent representation of the UI that is received and reasoned about by the fission to generate the interaction output.

The XML file describing the concrete UI is called **interaction output**. It stands for the modality-specific representation of the UI. The `ClientRuntime` receives the interaction output XML file via the MOM and interprets it to generate the UI at runtime. This way, the interaction output can be sent and realised on various devices in a distributed environment.

Listing 4.1 illustrates the structure of a simple interaction output file, which contains a text field and a picture bundled in a dialog act with a topic.

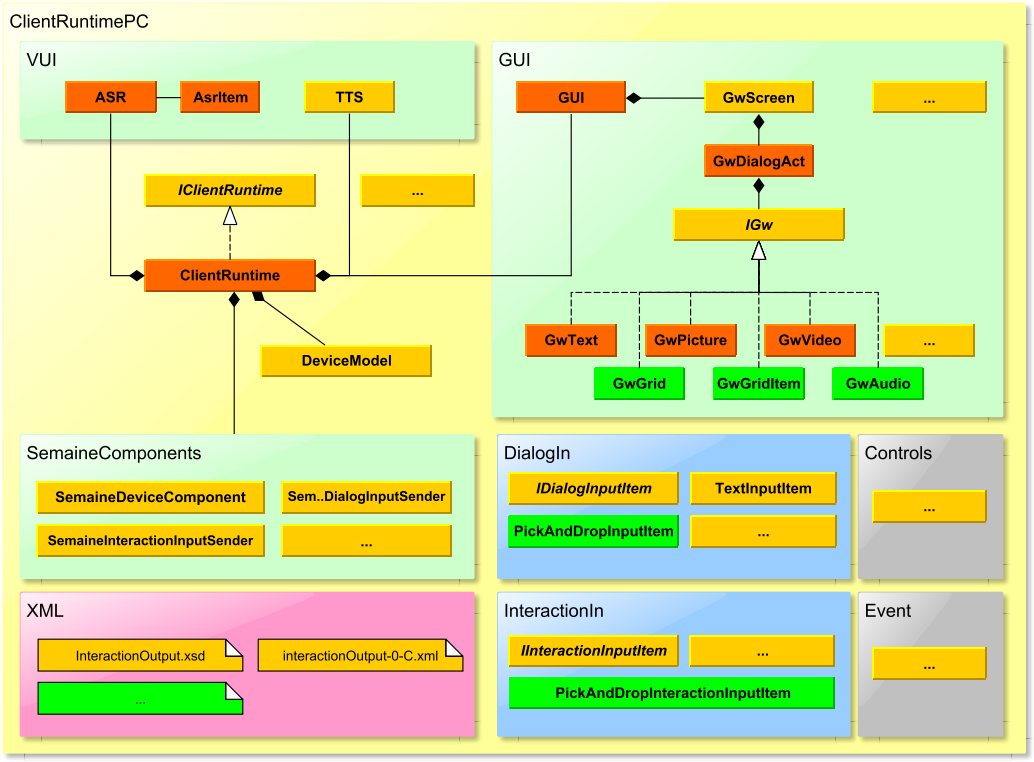

Figure 4.2: A reduced diagram that depicts the overall structure of the ClientRuntime. All components that have been modified for the prototype are coloured in orange-red and all components that have been added are coloured in green.

Dialog & Interaction Input

After inputs have been sensed and interpreted by the individual input devices and components, a modality-specific **interaction input** is created in the form of an XML file and sent to the fusion via the MOM. Until then, all inputs have been interpreted in isolation.

The fusion has the ability to combine inputs and creates an abstract, modality-independent **dialog input** XML file that is passed on to the dialog management via the MOM. The dialog management then handles the inputs accordingly and generates in turn the resulting outputs.

The dialog input represents the combined results mapped to a semantic representation by the fusion, as described in section 2.2.1 on page 14.

```
<?xml version="1.0" encoding="utf-8" ?>
<interactionOutput
    xmlns="http://sfb-trr-62.de/b3/InteractionOutput.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    initTime="2012-02-15T19:16:36"
    dialogOccurredTime="2012-06-04T08:41:16.059127+02:00"
    dialogID="MM-1.0">

    <outExpression deviceID="Experimental_Platform" deviceComponentID="TouchScreen">
        <topic>
            <text objectID="MM-1.0.topic" informationID="MM-1.0.topic">
                Metamorph Simple Pick-and-Drop
            </text>
        </topic>
        <dialogAct>
            <text objectID="textfield1" informationID="apple" isEditable="true"
                  nominator="Text 1:">
                Apple
            </text>
            <picture objectID="picture1" informationID="pineapple"
                     isPickable="true" isDroppable="true">
                data/images/fruits/pineapple.png
            </picture>
        </dialogAct>
    </outExpression>

</interactionOutput>
```

Listing 4.1: A typical interaction output XML file.

SEMAINE Components

The SEMAINE components are responsible for communicating with other components in the distributed systems, like other client runtimes or the dialog management, by sending topic-oriented messages like the dialog and interaction inputs or outputs. Multimedia controls like the audio and video player of the system also utilise their own media control sender to control the media players they are related to.

The Metamorph prototype uses these capabilities to send its inputs to the dialog management, although the dialog management is not modified to interpret and execute the Metamorph actions received.

4.1.2 Extending the Existing System

Although the given implementation already offers a lot of functionality, not everything that was required for the Metamorph interaction concept was available from the beginning. In

order to realise the use cases from section 3.2, a few additional components had to be added to the existing system, which will be explained shortly here.

Figure 4.2 also illustrates all components that have been extended or added to the existing system, with orange-red coloured components marking extension of existing components and green coloured components indicating newly added components.

Audio Support

On the basis of the already implemented `GwVideo` component and the WPF `MediaElement`, which already offers capabilities to play both audio and video media files, a base class called `GwMediaElement`, that combines the common aspects of audio and video, has been created. The movie control component was extended to support the audio player component. Therefore, it was renamed to media control. The type of media player is then set by an attribute called `controlType` in the interaction output.

Grid View

A grid view implementing the `IGw`-interface (see figure 4.2) was added to display a collection of information items in an orderly grid as part of the dialog act. The new grid view can be used to realise an abstract collection of information items. Figure 4.3 demonstrates a possible grid view.

Four classes had to be added: `GwGrid` and `GwGridItem` are located in the `ClientRuntime` assembly and handle the GUI-side to display the information on the concrete level. Backing classes called `Collection` and `CollectionItem` were added as well. They represent the simple concept of a collection and reside on the abstract level. The grid view is separated into two classes: The actual grid itself (`GwGrid` and `Collection`) and the items the grid contains (`GwGridItem` and `CollectionItem`, respectively), which in turn are composed out of existing text, picture and TTS text components.

Finally, the XML schema for the dialog output and interaction output had to be extended to offer tags for the collection on the abstract level and the grid on the concrete level. The code to generate those components from the XML interaction output files had to be added.

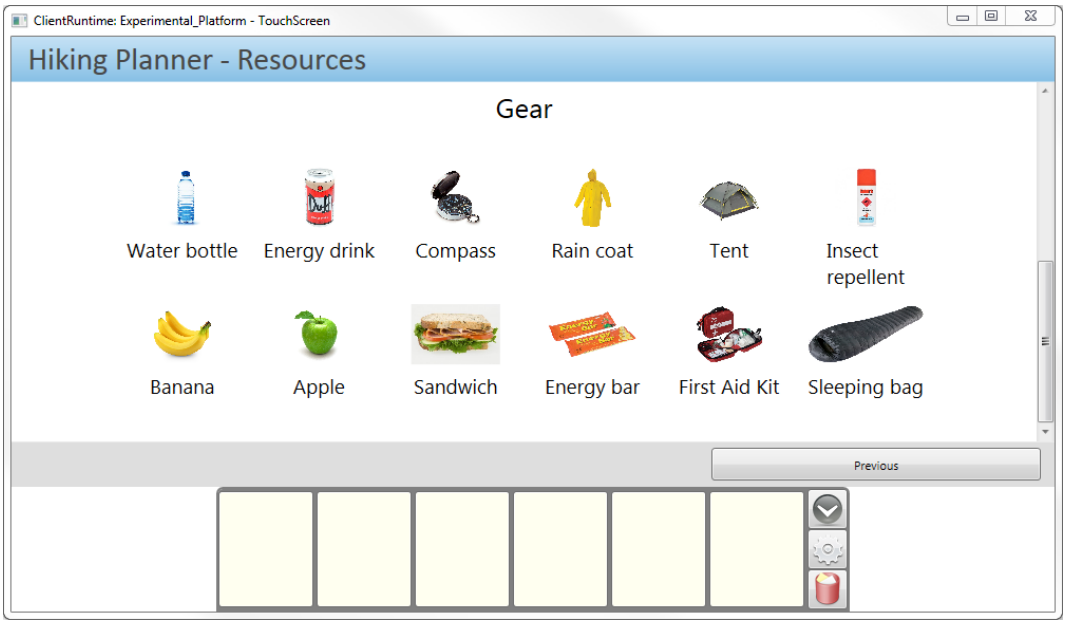

Figure 4.3: The grid view utilised in use case VI to position pickable gear in a well-arranged manner.

Mouse Gesture Support

Mouse gesture support was added to the `ClientRuntime` by utilising and adapting an existing library released under the *New BSD License* called *"Mouse Gestures for .NET"* [Kab10]. This is a simple library that records mouse movement when the right mouse button is held down. It differentiates between the four basic directions: Up, down, left and right. When the right mouse button is released, a `GestureRecognized` event is fired with `MouseGestureEventArgs` attached, which contains a `MouseGesture` object – an object that contains a recording of the gesture and other relevant information about the gesture.

The library was converted to .NET 4.0 (from .NET 2.0) and WPF-support was added by removing the tight coupling of the mouse gesture recognition code from the WinForms code and adding a WPF mouse filter – which is responsible for hooking up with the WPF mouse events and filtering out input that is used for gestures from the rest. Pre-existing mouse filters for WinForms and direct hooks into the Windows API were adjusted to use the new `MouseGestureRecognizer` class that now encapsulates the gesture recognition code.

Dialog & Interaction Input

The XML schemas for the dialog and interaction input files had to be extended by a new input type called `PickAndDrop`, which is shown in listing 4.2. As can be seen, the dialog input (as well as the interaction input) is extended by the basic actions as defined in section 3.3.1 on page 30. Each action is defined in the XML schema to contain the required data, as defined in the interaction concept, as well.

```
<xs:complexType name="PickAndDrop">
    <xs:choice minOccurs="1" maxOccurs="1">
        <xs:element name="pick" type="Pick"/>
        <xs:element name="drop" type="Drop"/>
        <xs:element name="select" type="Select"/>
        <xs:element name="deselect" type="Deselect"/>
        <xs:element name="clearSlot" type="ClearSlot"/>
        <xs:element name="clearUiComponent" type="ClearUiComponent"/>
    </xs:choice>
</xs:complexType>
```

Listing 4.2: The definition of the new PickAndDrop type for the dialog and interaction input.

4.2 System Design

This section elaborates on the system architecture designed to realise the Metamorph interaction concept. Two assemblies were added to the existing project. Those assemblies are called `Metamorph` and `MetamorphRuntime`.

The `Metamorph` assembly is a class library that contains all relevant code and functionality to realise the interaction concept on the abstract and concrete level. This includes code to store and synchronise information, UI components and backing classes for the clipboard, and lastly classes to interpret and execute the Pick-and-Drop interactions as defined in chapter 3. It is partitioned into four namespaces: `Sychronization`, `Storage`, `GUI` and `Interaction` (including the `Trigger` namespace). Figure 4.4 illustrates the namespace structure and the relationships between the classes. Each namespace is described in the following subsections.

The `MetamorphRuntime` is an executable that uses the `Metamorph` assembly and provides some global instances to the interaction concept.

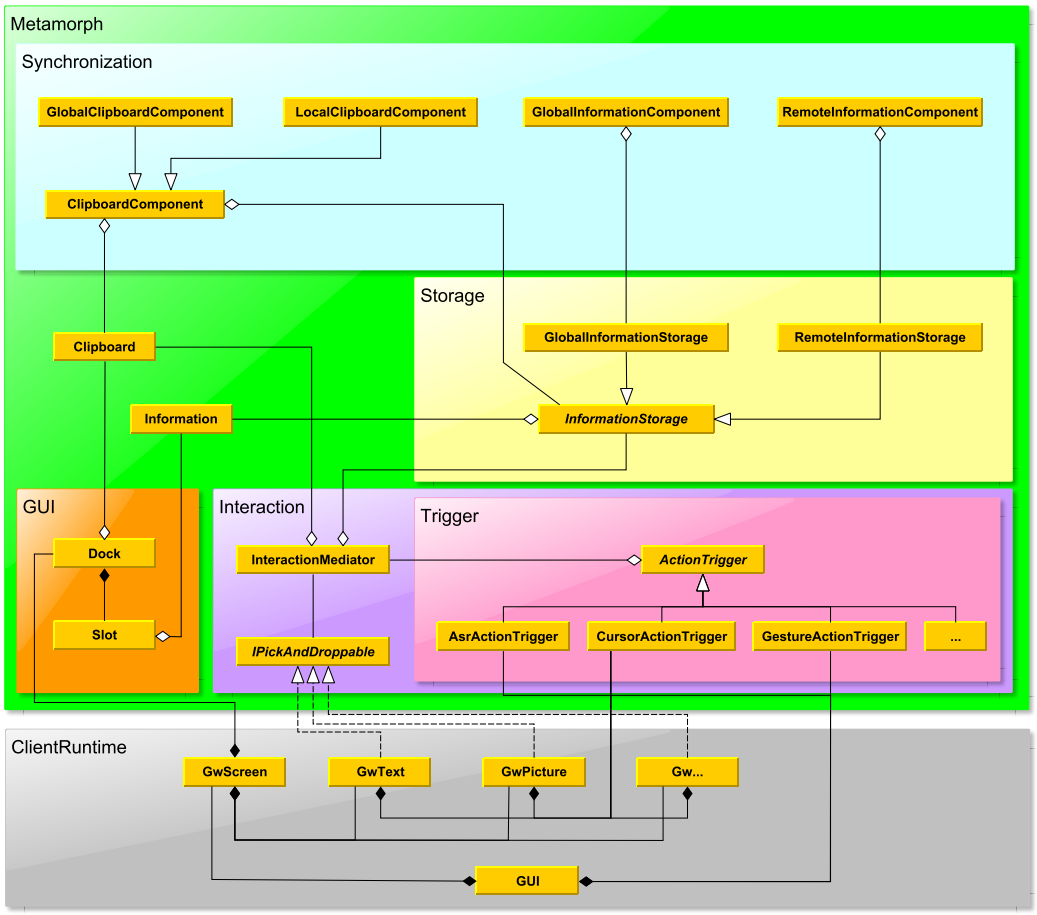

Figure 4.4: The overall namespace structure and class relationships of the Metamorph system architecture when used with the given ClientRuntime.

The `Clipboard` and `Information` classes are fundamental to the Metamorph interaction concept and therefore reside at the root of the assembly, in the namespace called `Metamorph`.

4.2.1 Synchronisation

The `Sychronization` namespace is responsible for the task its name stands for: Keeping everything synchronised across all clients in the distributed single-user environment. There are two hings that need to be kept synchronous for Metamorph: The clipboard and the information storage (which is described in more detail in section 4.2.2).

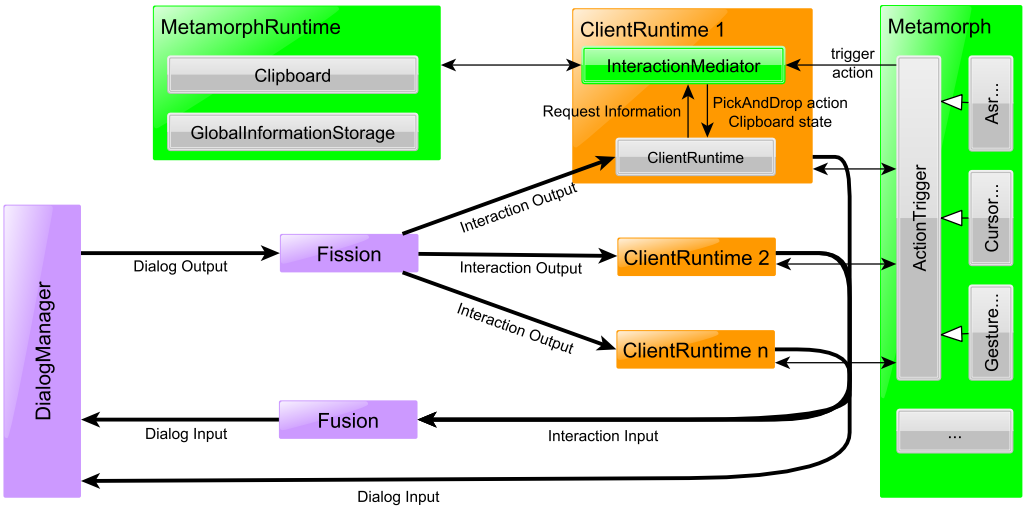

Figure 4.5: An overview of the communication paths and the integration of the Metamorph components into the existing system. The existing system's components are coloured purple, the ClientRuntime components are orange and the Metamorph components are green. Note that the Metamorph component actually belongs into each ClientRuntime like the InteractionMediator, but is shown outside to allow a better arrangement.

There exist two SEMAINE components for both cases: One responsible for the global part and one for the local part. There is only one global counterpart per system, started by the `MetamorphRuntime` as shown in figure 4.5, that performs tasks like initialisation or keeping the states of the clipboard and the information storage in sync across all clients.

A client can run on different or the same device as a separate UI component, which makes no difference to the synchronisation. It also allows the clients to dynamically connect or disconnect from the system at any given time without losing the clipboard state.

The clipboard components are derived from the common base class `ClipboardComponent`, since they are very similar and only deviate in small details – like which component is broadcasting updates and which one receives them remotely.

The `Sychronization` namespace only interacts with the clipboard and the information storages by holding references to them and subscribing to events to keep it loosely coupled.

4.2.2 Storage

The loading, storage and access to information is handled in the `Storage` namespace. The `InformationStorage` abstract class both is an interface and provides common functionality to the derived classes, which define how to load information and what storage technique to utilise, making the system more flexible and extensible. The base class also allows the other components to easily access the information storage by providing a static factory method.

Currently, there are two default implementations of information storages available: The `GlobalInformationStorage`, which loads the complete information data from XML into memory, and the `LocalInformationStorage`, which requests information via the corresponding SEMAINE component from the `GlobalInformationStorage` and caches the received information.

The information storages hold no references to the SEMAINE components and therefore effectively don't know about them. The SEMAINE components, instead, subscribe to the events of the information storages they are interested in.

4.2.3 Metamorph Runtime

The `MetamorphRuntime` is an executable assembly that is started once per system and takes on the role of a central data store. It starts the global variants of the clipboard and information storage classes as well as their related SEMAINE components.

It also offers a monitor depicted in figure 4.6 that registers and logs incoming and outgoing messages to the clipboard and information storage as well as general messages concering the startup and execution of the global components. This is achieved by subscribing to the events fired by the clipboard and information storage classes.

4.2.4 Interaction

Last but not least, is the `Interaction` namespace, which encapsulates input interpretation and action execution. The `InteractionMediator` is a mediator that sits between the input interpretation, done in the `Trigger` sub-namespace, and the backend of Metamorph, represented by the clipboard and information storage. The mediator abstracts away the clip-

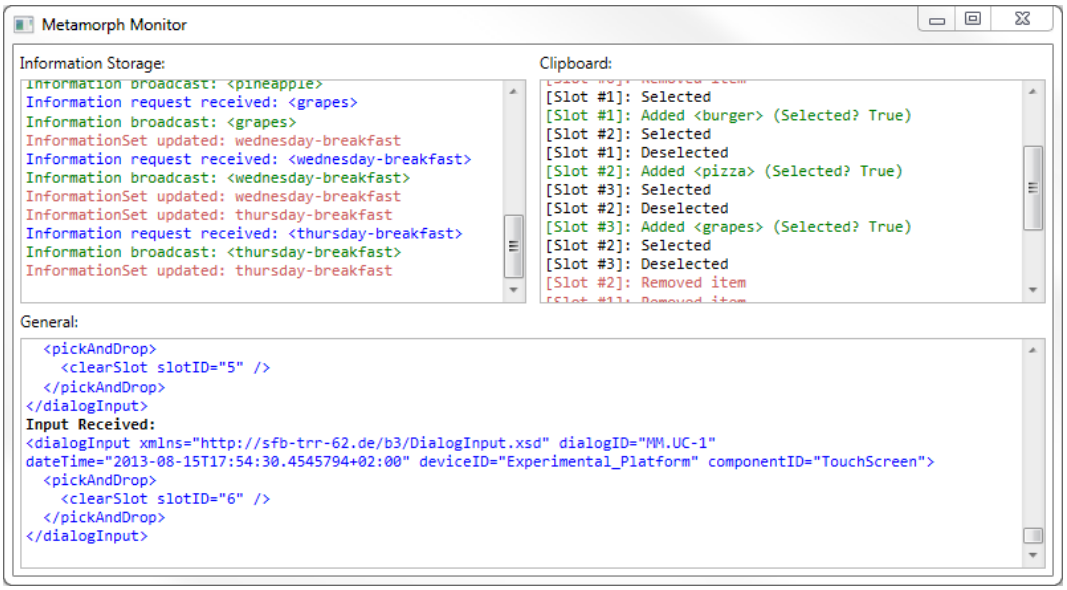

Figure 4.6: The monitor started by the MetamorphRuntime that tracks the messages exchanged via the MOM for the information storage and the clipboard.

board and information storage from the input interpretation and is responsible for handling the action requests either itself or by delegating certain tasks to the appropriate backend components where necessary.

This design was chosen for the prototypical implementation to short-circuit the dialog management, the fission and the fusion while still offering the ability to integrate them in a future work. The interaction mediator realises the abstract actions of the interaction concept and fires an event when one of the abstract actions is being triggered. These events are handled by the `ClientRuntime`, which in turn creates dialog and interaction input messages for those actions and sends them. Currently, these messages don't cause any effect.

The code for input interpretation can be found in the `Trigger` namespace as shown in figure 4.7. Every interpreter for a specific input modality – like ASR, cursor-based input methods (which stands for pointing devices like a mouse, touch screen, pen input, etc.) or gestures – is implemented in its own action trigger class. The base class `ActionTrigger` provides common functionality shared among all action triggers. But the actual input interpretation and triggering of actions is implemented well-encapsulated in the subclasses of the `ActionTrigger`. Common functionality, namely the actions to be executed when an input has been correctly identified, is bundled in the `InteractionMediator` class. The

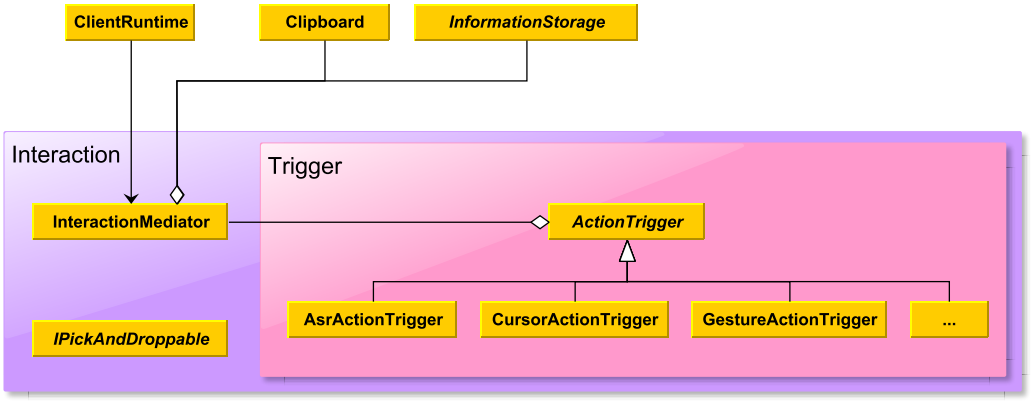

Figure 4.7: The ClientRuntime class references the mediator to subscribe to its events about actions being triggered to send the corresponding dialog and interaction inputs.

action triggers use the mediator to 'trigger' the actions when the matching input has been registered. This allows reusability of the abstract interactions and easy extensibility by new input modalities.

Since wiring the input modalities up with the UI is highly specific to the input modality itself, the action triggers are directly used by the UI components in the `ClientRuntime`. Generally, two kinds of input handling can be distinguished here: Global input handling (like ASR) and local input handling (like cursor-based input). Global input handling needs only to be instantiated in the base window of the UI and is then globally available. For example, a window either has speech recognition or not, it's not dependent on which UI component the user is interacting with. Local input handling, on the other hand, needs to be added to every single UI component that should support it and is then only locally available on those components. For example, drag & drop with a mouse needs to be implemented for the text field itself when dragging text from text fields should be supported.

4.2.5 Graphical User Interface (GUI)

The `GUI` namespace contains custom GUI components to be used in conjunction with the Metamorph interaction concept, like the dock. These components are loosely coupled with the `Clipboard` by subscribing to the clipboard's events that inform about an information being added or removed, a slot being selected, etc. This allows easy substitution and

extension of possible clipboard representations as the clipboard is clearly separated from its representation.

For this work, only the Metamorph dock was implemented as a proof-of-concept.

4.2.6 Voice User Interface (VUI)

A voice user interface could be added easily in the same manner that the GUI interface components were added as described in section 4.2.5 by subscribing to the clipboard's events. As with the GUI representations, multiple VUI components are possible following different approaches on how to represent the clipboard in an auditive way.

For this work, no VUI was implemented to keep the focus on the interaction concept.

4.2.7 Summary

The requirements for the interaction concept, defined in section 3.1.2 on page 25, derive to implementation specific requirements that need to be considered in order to achieve the goals of the interaction concept. A system has to be separated into well encapsulated and extensible components to support the multimodal input and output capabilities. Therefore, the system design is roughly separated into three components as defined by the model-view-controller design pattern.

The clipboard, being the central part of the interaction concept, is realised independently of any modality-specific representation on the abstract level and represents the model to fulfill the modality-independent nature of the interaction concept as defined in section 3.1.2. It is being synchronised by message-oriented middleware components that send and receive status updates concerning actions on the clipboard. This allows the clipboard to be used in a distributed system to allow the exchange of information across device borders.

An exchangeable visual dock is added on top of the clipboard to display its content and represents one possible kind of view on the clipboard model. The clipboard offers events that decouple the model from the actual components giving feedback to the user. For example, an auditive dock, serving as a voice user interface (VUI), could be easily added as an additional view due to this design. This allows the extension of the system by new output representations, making the clipboard independent of modality-specific feedback as stated in section 3.1.2.

An interaction mediator is added in-between the clipboard, action triggers and the UI components to serve as an encapsulating layer and central controller. The interaction mediator realises the modality-independent set of actions offered by the interaction concept (see section 3.3.1 on page 30), whereas the details for specific input modalities are hidden within their respective action trigger.

The action triggers are extensible implementations of specific input concepts, like automatic speech recognition (ASR). This allows the system to be extended by new input modalities without having to change the basic actions of the interaction concept.

4.3 Implementation

This section describes the overall implementation and highlights some noteworthy details.

Figure 4.8 illustrates the `Clipboard` class. It is a singleton that manages the clipboard's content by storing and providing access to information items. In addition, it offers events like `InformationAdded`, `SlotSelected`, etc. to inform other components – like the GUI dock – about state changes occurring in the clipboard. This allows a loose coupling between the abstract clipboard state – which is basically hidden from the user's perception without representation – and possible concrete representations of it. Adding a new kind of representation, like an aural dock, is therefore easily possible.

The `Clipboard` also keeps track of the most recently and least recently used information items. This is used to make smart choices when the clipboard is full and a new information item is added without a slot specified, as the oldest information item needs to be overwritten by the new one then. It also keeps track of the currently selected slot as this is a relevant information not only to the GUI, but to the interaction concept as well.

The `Information` class simply holds the data of an information item and has convenience properties to check which representations are available. It also features the possibility to create a dummy object without data which is updated later when it was retrieved through a request using the corresponding SEMAINE components (detailed in section 4.3.2). When an information object is updated, it fires an event called `InformationUpdated` to inform interested components about the change.

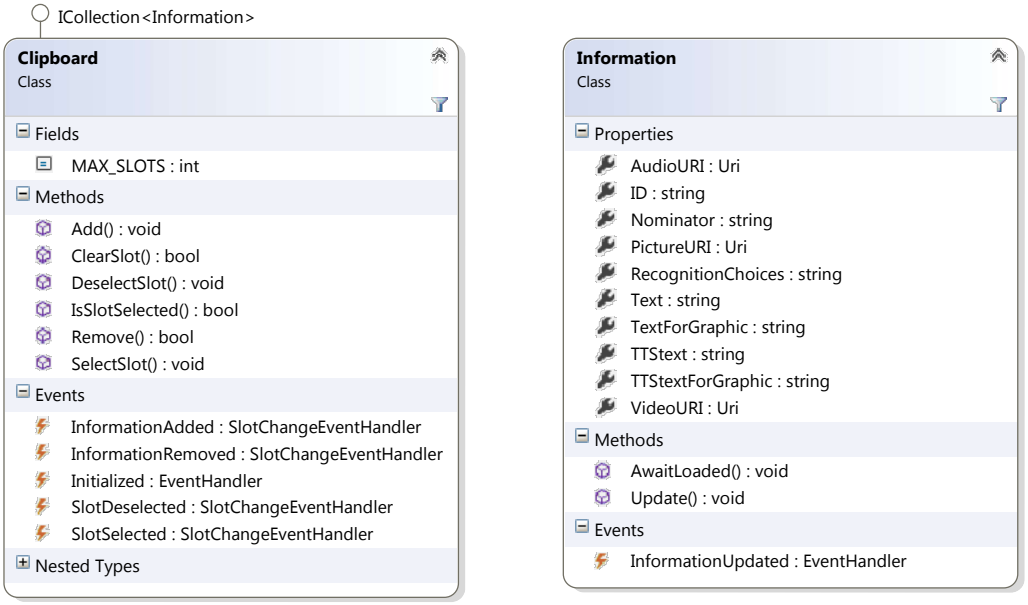

Figure 4.8: Condensed class diagram of the Clipboard and Information classes. See appendix A on page 103 for the full version.

4.3.1 Synchronisation

There are three classes related to the clipboard: `ClipboardComponent`, the abstract base class, `LocalClipboardComponent`, the component for the `ClientRuntime` (running locally on possibly remote devices), and `GlobalClipboardComponent`, the central component started by the `MetamorphRuntime`. A class diagram of this namespace can be found in appendix A on page 103.

The `LocalClipboardComponent` reports back changes applied on a `ClientRuntime` to the `GlobalClipboardComponent` running in the `MetamorphRuntime`. There can be any number of `ClientRuntimes` running on any number of (remote) devices in the system, but only one `MetamorphRuntime`. The `GlobalClipboardComponent` in turn broadcasts received updates to all local clipboard components, as illustrated in figure 4.9. Every local clipboard has an identical state as the global clipboard at all times and is kept in sync.

The `GlobalClipboardComponent` also sends its complete state on a separate topic exclusively used for initialisation requests when a new client connects. The initialisation request is sent by the `LocalClipboardComponent` once when it is started. After the initialisation is done, the local component unsubscribes from the initialisation topic.

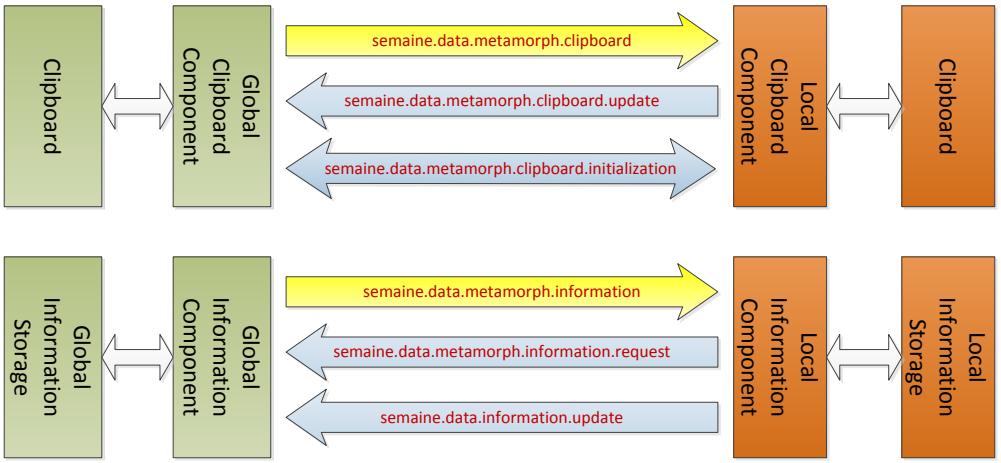

Figure 4.9: The communication and the MOM topics used by the synchronisation components. Broadcast topics are coloured yellow.

The `GlobalInformationComponent` stores and contains all information in the system. It acts like a central database. The `LocalInformationComponents` only request the information they need for their client at the time, caching the information they receive. They also send changes applied to the information they cache back to the global information component, which in turn broadcasts them to the other local information components so the information gets updated everywhere as soon as it is changed.

4.3.2 Storage

There are two implementations for the abstract `InformationStorage` base class: A global and a local version of an information storage. The `InformationStorage` base class itselfs offers a static factory method to instantiate its derived classes, so only a parameter has to be changed when a different implementation should be used.

The base class also uses the factory method to manage the lifecycle of the information storage objects by ensuring that only a single instance of any given combination of language and information storage implementation is instantiated at any time, effectively making the information storages singleton objects. This fact can be changed easily as the static factory method encapsulates the lifetime management.

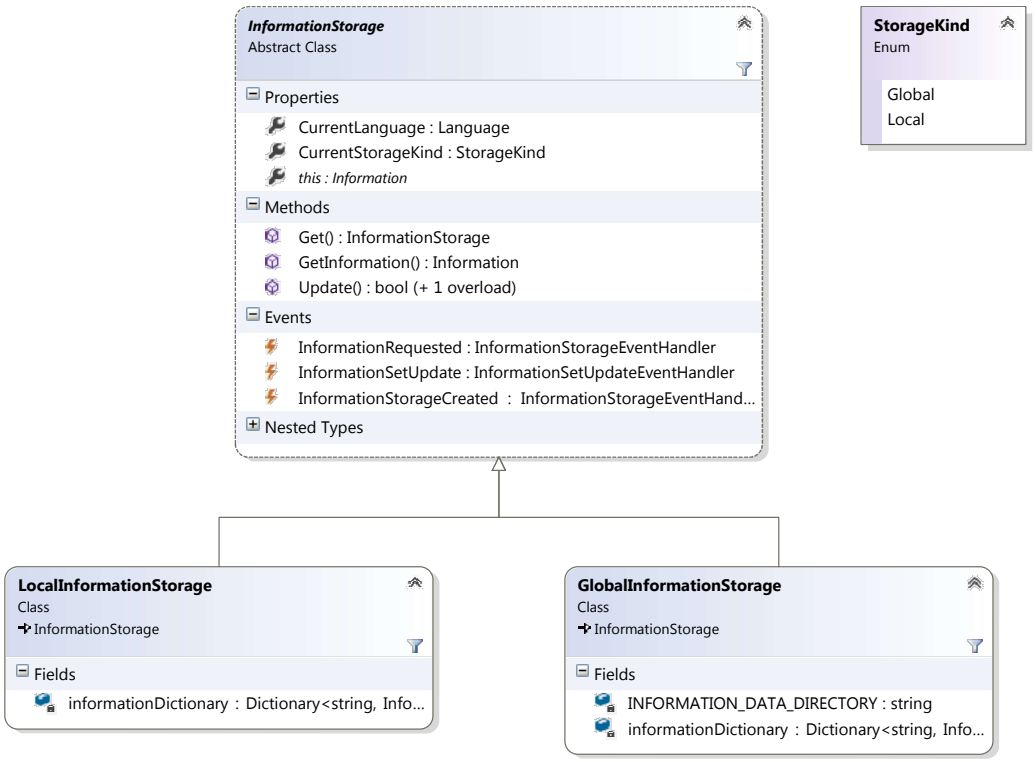

Figure 4.10: Condensed class diagram of the Storage namespace. See appendix A on page 103 for the full version.

The `GlobalInformationStorage` loads all information retrieved from an information set XML file into the memory and functions as the central information database. It is started by the `MetamorphRuntime` and responds to information requests sent by local information storages. Additionally, it also receives updates to the information it holds and allows the addition of new information. Updates to information are broadcasted to all `ClientRuntimes` that are connected with the global information storage via the MOM.

There already is a component called `InformationManager` in the existing system upon which the `GlobalInformationStorage` was based. It serves the same purpose as the global information storage introduced here and could be extended to replace the current `GlobalInformationStorage` altogether. The global information storage was created to have direct and full access to the information manager to allow adjustments necessary for the Metamorph interaction concept without unintentionally breaking the existing system's

functionality, as it was not clear at the beginning how much modification needed to be applied to the existing information manager.

The `LocalInformationStorage`, on the other hand, is empty when a new client is started and fills its cache just-in-time on-demand. This component was added in order to avoid delays introduced by requesting information using the MOM after the first request, while still retaining a central information repository. The caching strategy (or, if there should be caching at all) is up to the respective implementation. For the prototype, all requested information is cached indefinitely. This suffices for a proof-of-concept as the amount of information objects is expected to remain comparatively small for the use cases.

```
public override Information this[string infoId]
{
    get
    {
        if (!informationDictionary.ContainsKey(infoId))
        {
            // Add dummy object which is updated when the information has been retrieved
            Information information = new Information(infoId, isRemotelyLoaded: true);
            informationDictionary.Add(information.ID, information);

            onInformationRequested(
                new InformationStorageEventArgs(StorageKind.Remote, CurrentLanguage, infoId)
            );
        }

        return informationDictionary[infoId];
    }
    set { informationDictionary.Add(value.ID, value); }
}
```

Listing 4.3: The indexer for an information object in the LocalInformationStorage class.

As described in section 4.3, dummy information objects can be created with just an information ID not containing any data. This enables the `LocalInformationComponent` to send a request in the background to fetch the actual data from the global storage and later update the information object with the received data. It is necessary for the local information storage to return information objects that have not yet been cached or received without blocking as it cannot be said how long it takes to retrieve the information. See listing 4.3 for the indexer of the `LocalInformationStorage` that realises this behaviour. Notice that the `LocalInformationComponent` is informed about a request by a call to the method `onInformationRequested(...)`, which fires the respective event that the local information component subscribed to.

4.3.3 Interaction

The classes from the `Interaction` namespace represent the basic ingredients for the Metamorph interaction concept to work and are illustrated in the class diagrams in figure 4.11.

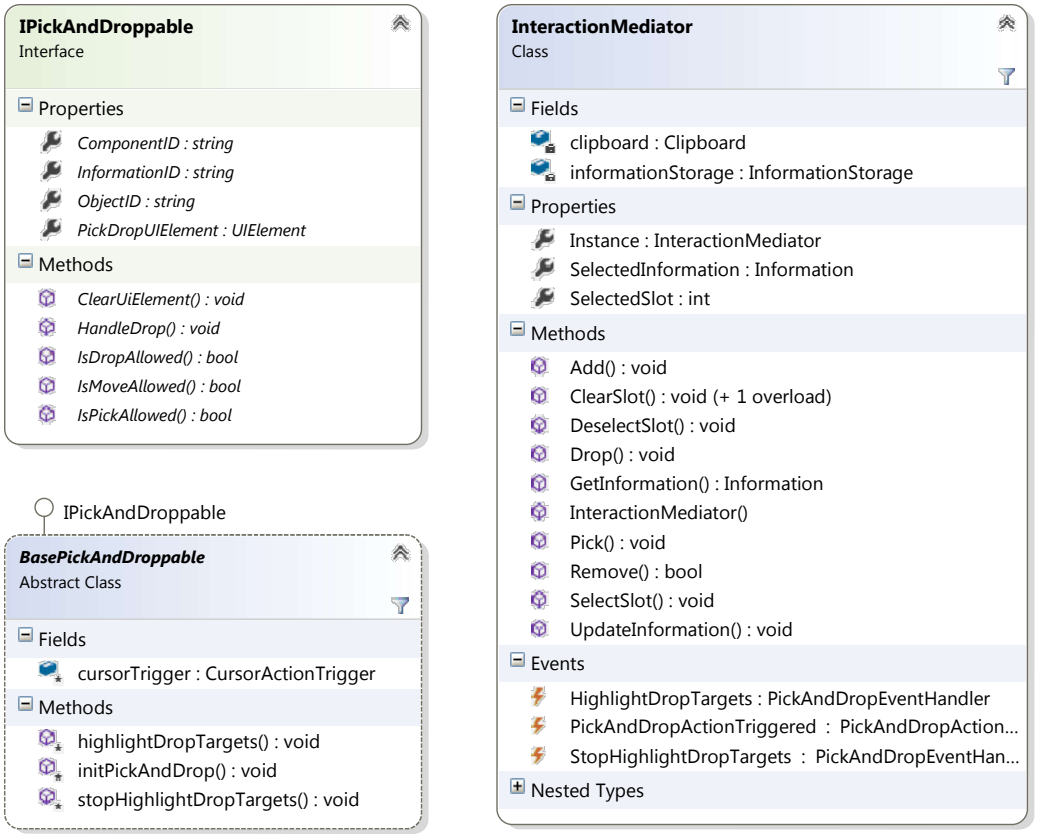

Figure 4.11: Condensed class diagram of the Interaction namespace. See appendix A on page 103 for the full version.

The `IPickAndDroppable` interface has to be implemented by all GUI components that should be pickable and droppable. It provides methods that indicate what is allowed and handle the operations that take place when a Pick-and-Drop interaction is being executed. The `InteractionMediator` and the various action triggers heavily rely on this interface to perform the necessary actions for a pick and drop operation without having to know any UI component-specific details.

```
protected void initPickAndDrop(bool isDroppable, string componentID)
{
    PickDropUIElement.AllowDrop = isDroppable;
    this.ComponentID = componentID;

    InteractionMediator.Instance.Add(this);
    cursorTrigger = new CursorActionTrigger(this);

    // Highlight GUI component when something is dragged that may be dropped here
    InteractionMediator.HighlightDropTargets += highlightDropTargets;
    InteractionMediator.StopHighlightDropTargets += stopHighlightDropTargets;
}
```

Listing 4.4: The standard initialisation of a pickable and droppable GUI component.

`BasePickAndDroppable` is an abstract base class that implements `IPickAndDroppable` and offers some common functionality like the initialisation of the Pick-and-Drop capabilities with a `CursorActionTrigger`, as shown in listing 4.4, which needs to be added locally to every GUI component (cursor-based interaction requires a graphical UI). It also subscribes to the `InteractionMediator` events used to highlight all GUI components that are valid drop targets during a drag operation and implements the default behaviour of adding a `DropShadowEffect` to the `PickDropUIElement` – which is the property defined in the `IPickAndDroppable` interface to identify the pickable and droppable GUI component.

The binding link between input interpretation and triggering the actions to execute is the `InteractionMediator`. It is a singleton object that holds references to the clipboard and the information storage. The action triggers reference the mediator and use it to trigger actions like picking up an item, dropping an item, retrieving information items, etc. The mediator also offers events to which all UI components subscribe in order to highlight them when an information item is being dragged that may be dropped on them – but only if the drop is valid after checking the available representations of the information.

For the prototype, the `InteractionMediator` executes the triggered actions and sends two kinds of input via the MOM: An `InteractionInput` and a `DialogInput`. These messages are received by the fusion and the `DialogManager` without causing an effect currently. When the system is fully adapted to the interaction concept, by modifying the fusion and the dialog manager, these messages will be interpreted and executed in those components instead. See section 4.4 on page 84 for more details on future work.

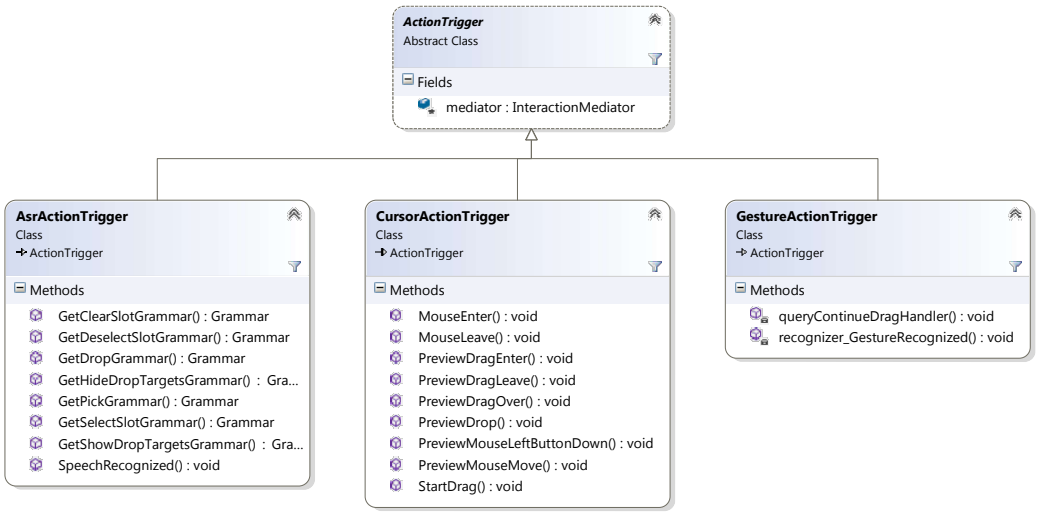

Figure 4.12: Condensed class diagram of the Trigger sub-namespace. See appendix A on page 103 for the full version.

Action Trigger

The action triggers encapsulate input modality-specific details and call methods on the `InteractionMediator` to act upon certain input triggers as defined by the Metamorph interaction concept in section 3.3.4 on page 43. The base class `ActionTrigger` simply offers access to the `InteractionMediator`, which is needed by all action triggers. Aside from that, the different input modalities have nothing in common and are highly specialised.

Three input concepts have been implemented for the prototype: Cursor-based inputs, speech input (ASR) and mouse gestures.

Cursor-based inputs by mouse, pen, stylus or touch input devices allow dragging and dropping in the traditional sense. This is implemented by subscribing to the drag & drop events provided by the WPF components and using the `DragDrop` class to perform the actual drag & drop. The pick action is triggered by dragging an information item over a dock slot and dropping it there.

ASR is implemented utilising the `SpeechRecognizerEngine` provided by the .NET Framework and already utilised by the existing system. The action trigger class simply provides building capabilities for the grammars according to the Metamorph interaction concept. These grammars are then loaded and handled by the existing ASR component

in the system, which has been extended to support Pick-and-Drop actions and call the `InteractionMediator` to execute them. For the prototype, the speech recognition is limited to the local UI component, since the fusion as well as the dialog manager would have to be modified to allow ASR to work across device borders.

The third action trigger implements **mouse gestures** using the approach explained in section 4.1.2 by subscribing to the `GestureRecognized` event and interpreting the mouse movements according to gestures defined by the Metamorph interaction concept.

4.3.4 The Dock Concept for Graphical User Interfaces

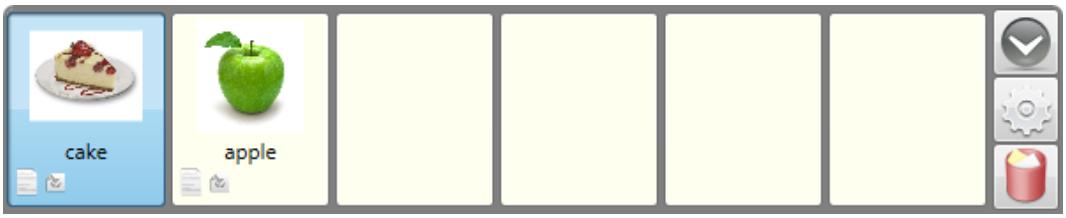

Figure 4.13: A Metamorph dock holding two information items with text and picture representations as rendered in the ClientRuntime.

The Metamorph `Dock` is depicted in figure 4.13. It is realised as a WPF `UserControl` that is composed of six individual slots. Every `Slot` is composed of existing WPF GUI components like a `ToggleButton`, a `Label` and the likes. Furthermore, the `Slot` class implements the `IPickAndDroppable` interface as introduced in section 4.3.3, meaning the slots themself handle Pick-and-Drop interactions. The `Dock`, however, subscribes to the clipboard's events and updates the slots when the information is updated or the clipboard's state changes (e.g. when an information item is removed on another client).

Extending the Dialog and Interaction Output for Pick-and-Drop

The schema for the interaction output needed to be extended by the abstract clipboard and new attributes on all pickable and droppable UI components to indicate their level of support for Pick-and-Drop. Similar additions have been made to the dialog output schema on the abstract UI level.

Listing 4.5 illustrates the addition of the abstract clipboard to the interaction output schema as a component at the root level of the interaction output. The slots are given as sub-tags with their ID, selection state and any information they hold as value.

```
<!-- the abstract clipboard -->
<xs:complexType name="Clipboard">
    <xs:sequence>
        <xs:element name="slot" type="Slot" minOccurs="6" maxOccurs="6" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="Slot">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="id" type="xs:integer" use="required" />
            <xs:attribute name="selected" type="xs:boolean" use="optional" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

Listing 4.5: The schema extension in the InteractionOutput for an abstract clipboard.

Attributes called `isPickable`, `isDroppable` and `overrideNominator` are shown in figure 4.6 where they extend the text UI component exemplarily. The first two are used to determine if picking or dropping is allowed on the respective UI component, whereas the last determines if the nominator of the UI component shall change upon a drop.

```
<!-- the basic types -->
<xs:complexType name="Text">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="objectID" type="xs:string" use="required" />
            <xs:attribute name="informationID" type="xs:string" use="required" />
            <!-- ... -->
            <!-- Start Metamorph -->
            <xs:attribute name="isPickable" type="xs:boolean" use="optional" default="false"/>
            <xs:attribute name="isDroppable" type="xs:boolean" use="optional" default="false"/>
            <xs:attribute name="overrideNominator" type="xs:boolean" use="optional"
                          default="false"/>
            <!-- End Metamorph -->
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

Listing 4.6: The InteractionOutput schema that defines a text component on the concrete level.

The new attributes can then be added to the interaction output. Listing 4.7 shows an example interaction output extended by the Metamorph interaction concept (the Metamorph additions have been marked yellow).

```
<interactionOutput dialogID="MM-1.0">

    <outExpression deviceID="Experimental_Platform" deviceComponentID="TouchScreen"
                   hasClipboardRepresentation="true">
        <dialogAct>
            <text objectID="textfield1" informationID="apple" isEditable="true"
                  isPickable="true" isDroppable="true" overrideNominator="true"
                  nominator="Text 1:">
                Apple
            </text>
        </dialogAct>
    </outExpression>

    <clipboard>
        <slot id="1" />
        <slot id="2" />
        <slot id="3" />
        <slot id="4" />
        <slot id="5" />
        <slot id="6" />
    </clipboard>

</interactionOutput>
```

Listing 4.7: The interaction output from section 4.1.1 extended by the Pick-and-Drop attributes.

First of all, the attribute called `hasClipboardRepresentation` has been added to the `outExpression` in listing 4.7. It indicates whether the given concrete output expression shall have a concrete clipboard representation, like the GUI dock for visual outputs or an auditive dock for aural outputs. Second, the text UI component has been extended by the Pick-and-Drop attributes defined by Metamorph to indicate that the text UI component should be pickable and droppable elements in the UI, with the nominator adapting to the dropped content.

The backing classes were extended by corresponding properties for the new attributes and are initialised from the interaction output XML. These backing classes are then queried by the UI component classes like `GwText` to check if pick or drop is allowed. This enables an adaptive component behaviour according to the interaction concept.

4.4 Interim Conclusion

This section concludes the prototypical implementation of the Metamorph interaction concept. It has been shown how the interaction concept was translated into a system design and realised in the existing system in the output and input device components. Future prospects concerning the prototype are described in section 6 on page 102.

The prototype took the requirements described in section 3.1.2 on page 25 into consideration: A pick & drop-like interaction was realised that allows the handling of multimodal information items, which are stored in an information storage that is globally available in the distributed system. The information items transition into the matching representation based on the UI components they are dropped upon, thus taking the context into account.

A global clipboard that is synchronised across devices allows the storage of multiple information items. The user can cross device borders and execute other actions after an information item has been picked up and stored in the clipboard. He is also free to change input modalities while working with the Metamorph interaction concept at any given time.

Feedback has been implemented only for the visual channel in the prototype, but was designed to be decoupled from specific views. It is possible to add auditive feedback to indicate drop targets or add additional views to make the clipboard visible to the user. This can be achieved by utilising different output modalities or a combination of output modalities.

5 Evaluation

This chapter details the expert evaluation that has been conducted to gain further insight and collect feedback on the effectiveness, efficiency, learnability and usability of the interaction concept developed.

After the prototype has been implemented, the expert evaluation was conducted and is described in the following sections. In this evaluation, several experts have been interviewed on their impressions and opinions about the interaction concept demonstrated using the prototypical implementation. This was arranged to study the expectations and reactions of the experts concerning the presented interaction concept. The evaluation was done during the course of one week and took place in a separate room at the University of Ulm.

5.1 Participants

A total of five experts, all male, between the ages of 28 and 33 (with an average of 30) have been interviewed to gain further insight and feedback concerning the interaction concept and its prototype. According to Nielsen [Nie00], five participants provide sufficient results for a first evaluation iteration.

These five experts came from the fields of human-computer interaction, ubiquitous computing, multimodal interaction, distributed systems and speech interaction. All participants were research associates recruited from the University of Ulm. Only one of them had prior experience with the existing system.

5.2 Test Setup

The tests were conducted on a single computer utilising two monitors to simulate two different devices. The kinds of devices were indicated by small stand-up displays.

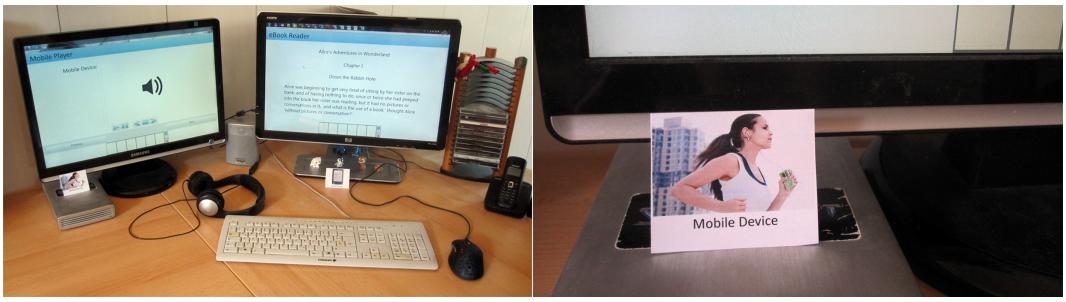

Figure 5.1: The computer setup for the expert evaluation. Small paper stand-up displays (see the picture on the right) were used to indicate the actual devices intended to be used by use case V.

Figure 5.1 shows the test setup and the stand-up displays used to indicate the devices used in the use cases. Up to two instances of the client runtime were used to demonstrate the use cases detailed in section 3.2 on page 27. Speech input, mouse and mouse gestures were available as input concepts. A microphone was used to record the interview with the experts. Furthermore, each expert was asked to fill out a questionnaire.

5.3 Procedure

At the beginning of each interview, the expert had to fill out the first page of the questionnaire, which gathered some personal facts like age, academic degree, the expert's field of expertise, his competence in various knowledge fields related to this work and an agreement that allowed using the interview in this work.

Afterwards, the use cases from section 3.2[1] were explained and demonstrated to the expert, one by one. The experts were encouraged to state remarks whenever they wanted to. For each use case, some questions were asked to learn more about the expert's opinion on some specific aspects demonstrated by the use case.

Lastly, the expert was asked what he regarded as the most important aspects to consider when developing a multimodal interaction concept like the one presented in the evaluation. Additionally, the expert was asked to rate some aspects of the interaction concept in the questionnaire and to extend an XML example of an interaction output by marking an UI component as pickable and droppable and to extend it by a clipboard.

[1] With the exception of use case III, as its main difference lies in the devices used, which were not available for the prototype. So no new insight would be gained by this use case.

5.4 Statements of the Experts

The feedback that has been received by the experts is summarised in this section, starting with the overall ratings the experts attributed the prototype. Figure 5.2 depicts the average ratings collected by the questionnaire for the general usefulness of the use cases, as well as the consistency, efficiency, effectivity and the learnability of the prototype (with 1 representing “bad” and 5 “very good”).

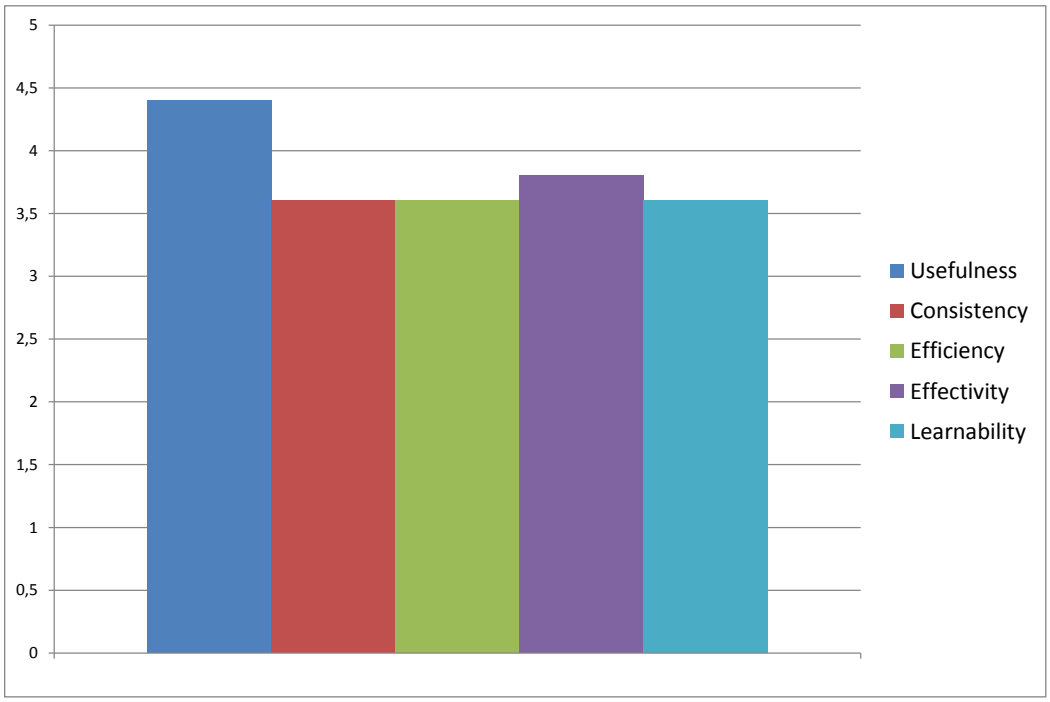

Figure 5.2: The average ratings attributed to several aspects of the interaction concept.

The usefulness was rated high with an average rating of 4.4 out of 5, confirming the motivation for this work to provide a multimodal Pick-and-Drop-like interaction concept across devices. Consistency, efficiency and learnability have all been rated above the average with a rating of 3.6 out of 5, but there is clearly potential for improvement here. Finally, the effectivity was rated slightly higher with a rating of 3.8, which also leaves some room for improvements, but shows that what can be achieved with the interaction concept was regarded generally valuable. The following sections will go into detail on the feedback the experts gave, providing insight into and reasons for these ratings.

Overall, however, these ratings are a reassuring, positive feedback for the interaction concept, proving that it has the potential to contribute to a better user experience and offers beneficial interaction options.

Based on the experts' statements, eight aspects for discussion are highlighted in the remainder of this section.

5.4.1 Visibility

The visibility of the interaction options and feedback to show what result has been accomplished by the user was considered a very important aspect during the evaluation by the experts. This affected the ratings for learnability.

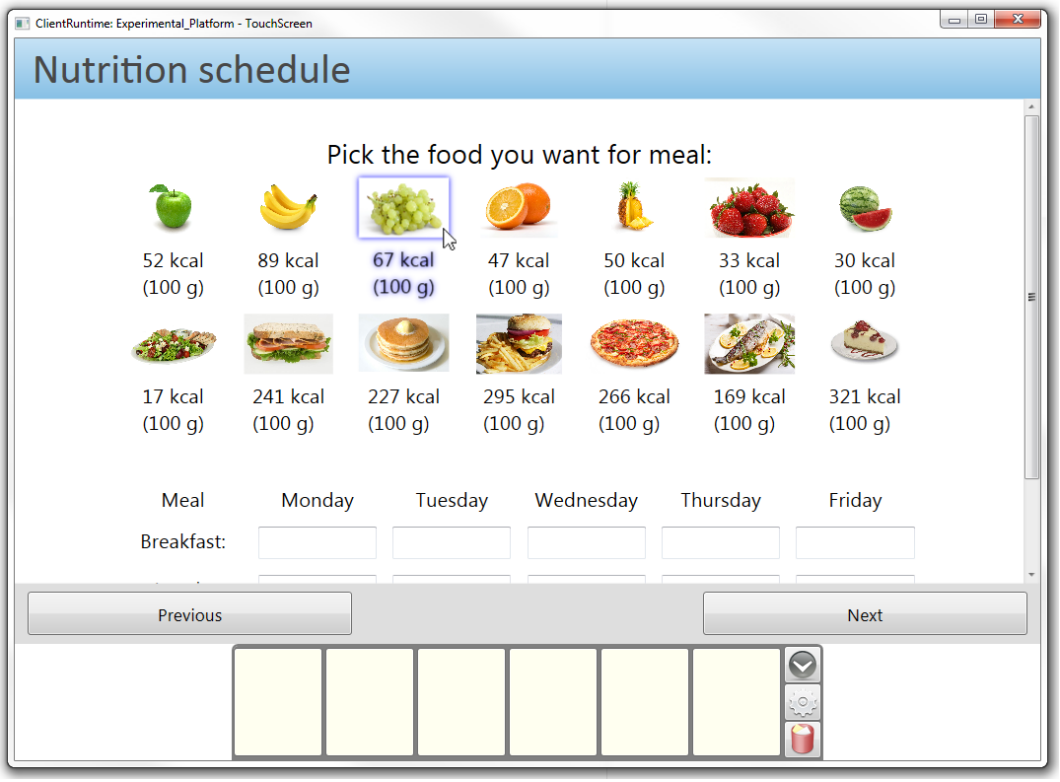

Figure 5.3: The prototype showing a grid view from use case II where the user can pick food. Note the blue glow highlighting the item under the mouse cursor.

When being asked how to indicate pickable or droppable UI components to the user, all five experts stated highlighting the UI component with a coloured border as a possible solution.

This affirms the chosen solution with glowing borders as shown in figures 5.3 and 5.4. All five experts considered the highlighting solution, as realised by the prototype, to be good. One expert noted that it would be more convenient to highlight pickable UI components all the time and not only on user demand (e.g., by hovering with the cursor over the UI component as shown in figure 5.3). Having a subtle highlight that is always visible would avoid that the user has to search for pickable information items.

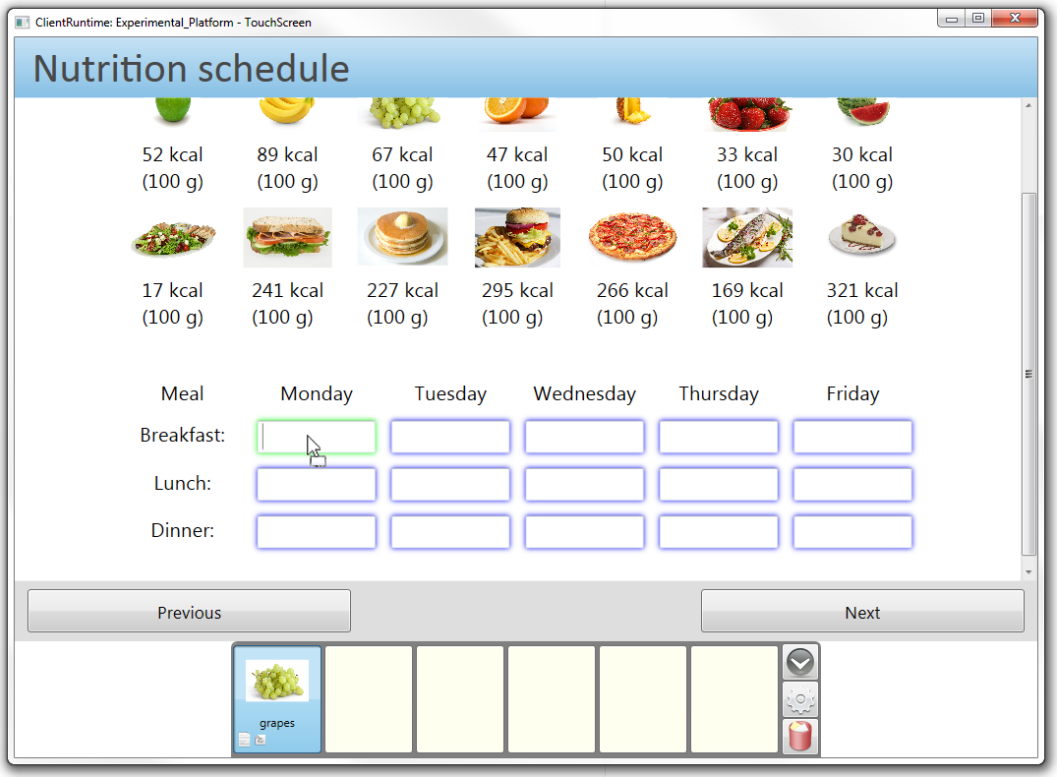

Figure 5.4: The prototype showing a nutrition schedule. Note the highlighting of valid drop targets (blue glow) and the highlighting of the GUI component the mouse cursor hovers over (green glow) to indicate the currently targeted drop location.

Four experts emphasised the necessity to encourage the user to “grab and move” the pickable UI components – in contrast to simply clicking/tapping them – when using cursor input. One expert stated that graphical objects customarily encourage a user to move them, consequently using some kind of graphic to indicate movability would follow known conventions.

Another expert suggested playing an animation on cursor-over, which implies that the UI component is “loose” (and thus movable), to encourage users to drag the UI component. Lastly, it was noted that it is very important to clearly distinguish pickable text from hyperlinks to prevent user confusion.

Using symbols to indicate possible representations on pickable or droppable GUI components was suggested by four experts, supposable on a cursor-over to avoid cluttering the GUI. This would visibly communicate the available representations on pickable components and the transition that will happen on a drop. The latter would help the user to understand and predict the system’s behaviour better, which is a desirable attribute [Nor02].

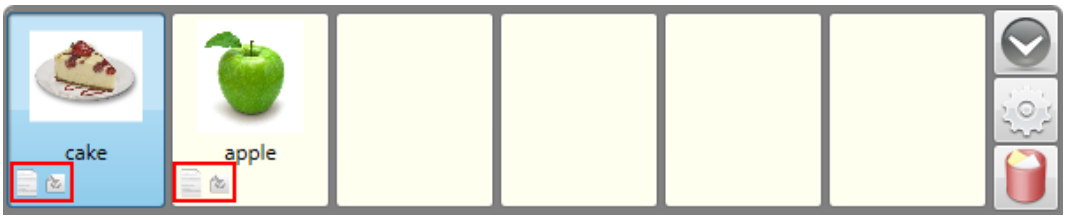

Figure 5.5: The symbols used to indicate the available representations in the dock.

The symbols in the dock below the information item’s name serve a similar purpose (see figure 5.5), but they are limited to the dock. The experts stated having the symbols appear near the pickable or droppable GUI component would be more convenient. Furthermore, one expert noted that the symbols in the slot are only visible when the user explicitly picks an information up using the dock, but not when directly dragging and dropping an item without using the dock. Showing a preview of the currently dragged item in a dock slot during a drag was suggested. This might be a suggestion worth implementing, as it makes the possibility to drop something into the dock clearly visible. Also, having a preview that visualises what will happen when the user drops an item into a slot, again helps to understand and predict the system’s behaviour. One expert suggested having an icon preview directly at the cursor while dragging. This could be enhanced with small icons for representations below the information item’s icon as long as this does not confuse the user [WCO95].

Three experts suggested highlighting the whole area of the drop target (e.g. with a specific background colour and/or pattern). The practicability of this suggestion is constrained by the respective GUI component: While it may be sensible to change the background colour of a text component, it’s not effective to do the same for a picture or video component, as this would overlap the content.

5.4.2 Multimodality

Having multiple input modalities at one's disposal was overall well received by the experts. On the one hand, four experts found it to be generally advantageous, as it offers each user the freedom to interact with the application in the user's preferred way. This confirms the hypothesis from chapter 3, that users prefer to communicate multimodally.

On the other hand, three experts also noted that the kinds of modalities that make sense are application and device-specific. For example, a typical user might expect and use gestures on touch devices like a tablet, but not on a desktop computer utilising a mouse. Additionally, the context needs to be considered as well. The option to deactivate certain modalities is necessary for cases where they might be improper due to environmental or security reasons.

When offering many different input modalities, three experts found the need to indicate in some way what input concepts are available to the user at any given time. Because, obviously, when the user has a fragmentary mental model of the application and does not know or expect the available input modalities, he cannot utilise them. The user may even be confused by the system's behaviour, for example when using the system while speaking out loud or with a colleague about what he is doing, accidentally triggering a speech command.

One expert expressed that a combination of modalities is favourable, like combining speech commands with body gestures in a "Put That There"-style of interaction. This was touched upon in section 3.3.6 on page 48, where it was ascertained that a combination of modalities is in some cases even necessary to complement the expressiveness of the diverse input modalities.

A very interesting suggestion was made by one expert concerning the decoupling of the abstract interaction concept from the modality-specific mappings: He suggested using a model-based approach for the input modality mappings to the Metamorph actions. When realising speech input, the actual words or phrases used to trigger the actions could be specified in the model files. These model files could then be changed and dynamically reloaded at runtime to allow adaptive speech recognition. This would allow, for example, the easy addition of a natural language interface to the speech recognition or localised speech interfaces.

However, the feasibility of this approach has to be verified first, as it would need a very powerful and flexible framework for modelling the many different input modalities like mouse,

keyboard, gestures, speech, etc. This approach runs the risk that the framework needs to implement most of the input modality mapping specifics so that the flexibility of having a model-based approach is very limited.

5.4.3 Transient Media Types

The majority of experts (three out of five) saw no problems when transitioning from a static information representation to a transient representation as long as it is a conscious decision made by the user. In this case, it was argued that the user knows what to expect since he should be familiar with the audio and video media types – even though he may not be familiar with these media types in the context of transitioning representations.

One expert stated that it was necessary to offer options to navigate the transient media, which is already provided by the system. He also suggested allowing the possibility to show several representations at the same time, side by side, like having a text instruction next to the video instruction for reference. This enhances the multimodal aspects of the concept and might be worth investigating.

All experts agreed that a cut operation in a distributed system should be safeguarded by a warning dialog because there might be devices that do not offer storage space for the information data, which could lead to a potential data loss. Another proposed solution was to centrally store the information in a cloud-like storage and to never delete it from there, even when the information is deleted from every single device in the distributed system. This is the current approach by the prototype.

Disallowing the cut operation was rejected by two experts, stating that it would be inconvenient when the user has to copy and delete an information manually when he wants to move it. This argument is reinforced in a distributed system with many devices, which would multiply the manual workload by the number of devices affected. Another expert suggested having a history of all actions, allowing the user to undo them as a safeguard.

One expert expressed a concern about content mismatch between different representations, like a text deviating from the spoken audio representation of the text. This violates the user's expectation to get exactly the same information in another representation and should be prevented. Even though this is a valid concern, it does not affect the interaction concept itself.

5.4.4 Hierarchical Information

For hierarchical information, four experts said they expect the system to remember and jump to the current position of the information even when it is transitioned into another representation. Three experts said that the current position should be automatically synchronised with the whole system. Two experts even suggested using an eye tracker for information items like an eBook to capture the current paragraph or sentence the user was last looking at – noted however that this is not trivially realisable. One expert had the idea to support user-defined bookmarks for hierarchical information items, increasing the convenience of the system. Overall, the majority of experts rated the need to address and jump to positions inside of hierarchical information as very important.

When transferring hierarchical information to other devices, like a mobile device, two experts stated they expect the system to transfer the complete information by default. Additionally, however, the user should have the option to transfer only selected parts by the pick operation due to space constraints. One expert noted that this should be the default behaviour when the user explicitly picks a subset of the information, drawing a comparison to dragging a selected text in a word processor. This should be investigated in a usability study.

Having a mismatch of the information structure for different representations was noted as a problem by three experts. Different representations might offer different levels to reference, like a book being divided into chapters, sections, paragraphs and sentences, whereas movies are most often only divided into chapters. This mismatch is inherent to the different representations and is not trivially solvable. Very good semantic annotations for all representations would be needed to handle this adequately, but one expert raised the question of the benefit for such a detailed match.

5.4.5 Dynamic Nominators

The concept of dynamic nominators received mixed reactions, with three experts being intrigued by the idea, one stating that its usefulness depends on the application. All three agreed that there are cases where it might be meaningful to add a higher level semantic to the content through a dynamic nominator. Two experts emphasised the importance of semantic validation, so the content fits the context. For example, dropping a location into a list of food should be disallowed for the concept to work without seeming out of place.

Two experts expressed the need for an adequate presentation of the feature, one stating that visual labels should not be used as dynamic nominators, since the user expectation is that labels do not change. Instead, the dynamic nominator should be presented alongside the content to indicate its close relation to it. Both experts suggested using a graphical presentation for this concept in use case VI, having a graphical backpack with individual items being categorised by the dynamic nominators inside the backpack. This is also an example for the need to support hierarchical information structures.

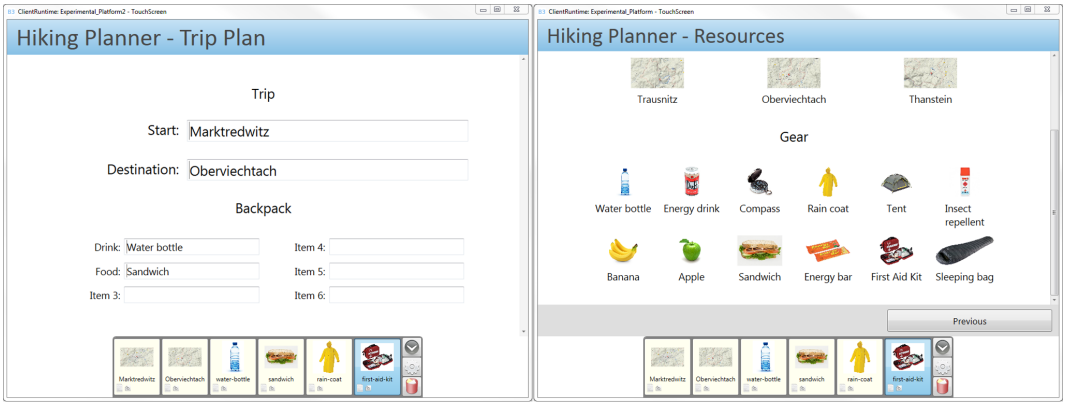

Figure 5.6: Use case VI as modelled in the prototype. The left side shows the trip planner interface on a computer whereas the right side is supposed to show the locations and gear on a tablet. Notice that the labels in the backpack on the left side are dynamically adapted to its content.

During use case VI, which showcased the dynamic nominators, four experts also noted that the information items should be directly referenceable using the speech interaction (e.g. “Pick up *apple*” instead of “Pick up *item 1*”). They argued that this is more intuitive and feels more natural to the user. This also remedies the possible problem of referencing the UI components through their labels, when the label changes on drop actions – as it does in the prototype (see figure 5.6).

To enable this, the speech input would have to be extended to identify UI components based on their contents, with ambiguities being resolved by standard mechanisms used in today’s speech interfaces (usually by presenting a list of alternatives). Three experts also said that the UI components should be referenceable by their position, like “the first text field under trip”.

5.4.6 Interaction

Concerning the interactions in the concept, two experts pointed out that being able to directly pick & drop an information without using the dock is of significant importance for user convenience and efficiency. This possibility was already mentioned in section 3.3.1 on page 30, but was not implemented in the prototype except for pick & drop with a mouse.

Providing feedback on the same communication channel that was used for input was suggested by another expert. In other words, the system should give auditive feedback to speech input and visual feedback to cursor-based inputs. This is another case that would have to be determined by extensive usability studies with end users to determine its impact.

One expert found the swipe gesture to the top confusing when the dock is located at the bottom of the screen, as it seemed illogical to him. His natural mapping of the gesture was not that of picking something “up”, but rather to “throw” something into the dock for storage. It needs to be evaluated, with a statistically significant amount of end users, what interpretations there are amongst the end users and how they are distributed.

Another important point to consider was the ability to directly drop an information item onto a device, rather than an UI component. Two experts argued that this is what the user wants to achieve eventually in many cases. It is also more convenient when dropping an information item on devices with a single main functionality, like an mp3 player. Using the dock or clipboard is, after all, only a means to an end. A default information representation (and potentially a default program for more complex devices) to use has to be defined for each device in this case.

5.4.7 Generally Important Aspects

At the end of the evaluation, all experts were asked what they considered the most important aspects to consider when devising a multimodal interaction concept that works across device borders.

Being able to drop an information directly onto a device is closely related to what one expert considered an important aspect: That everything has to work seamlessly. Allowing to drop something onto a device, like a wall-mounted touch screen, is closer to what the user wants to achieve in many cases. It feels more natural and minimises the interactions necessary to achieve the user’s goal, which is another aspect of a seamless interaction concept.

Two experts stated that a good interaction concept supports the user in achieving his goals. One expert emphasised that usability studies with end users are a crucially important part to come up with an interaction concept that matches the user's needs. For example, it is important to know the vocabulary most users naturally use to create a speech recognition which is intuitive to use. The same applies for mouse, touch and body gestures.

The modalities for input and output need to fit the use case as well, which was already touched upon in the last section for input modalities: The environment and context needs to be taken into consideration when determining which input modalities make sense under the given circumstances. For output modalities, this means that the information needs to be modelled in a way that supports the use cases. Another expert said that the information should be semantically annotated and linked in a manner that is suitable for the use case, which is more of a technical problem that contributes to the usability of the interaction concept. Semantical annotations of the information were mentioned several times and would enhance the interaction concept in a meaningful way since it constrains the user's interaction possibilities by prohibiting invalid actions [Nor02].

5.4.8 Extending the Interaction Output

The experts where asked how they would extend a simplified interaction output example in XML, which is shown in figure 5.7. This question was aimed to uncover problems potential developers may have when working with the XML extensions introduced by the interaction concept. Four experts added two separate attributes, one for pick and one for drop, to the text components. This approach matches the pickable and droppable marker introduced in section 3.3.1 on page 30.

Interestingly, three experts asked at first whether the UI components should have to be marked at all. They suggested that all UI components should always be pickable and droppable, but realised after some thought that the pickable and droppable elements of a user interface are rather application-specific. Still, it is curious that three out of five experts intuitively had the same mindset at first.

One expert then came up with the idea to automatically deduce which UI components should be pickable and/or droppable, based on a semantic description of the UI or by extracting available metadata from existing systems. The expert named YouTube as an example: The system could determine that the video is an information that is pickable

```
<interactionOutput dialogID="example_dialog">

  <outExpression deviceID="PC" deviceComponentID="TouchScreen">
    <dialogAct>
      <text objectID="text" informationID="info1" isEditable="true"
            nominator="Text 1:">Apple</text>
    </dialogAct>
  </outExpression>

  <outExpression deviceID="Smartphone" deviceComponentID="SpeechInput">
    <dialogAct>
      <text objectID="text1" informationID="info1" isEditable="true" nominator="Text 1" />
    </dialogAct>
  </outExpression>

</interactionOutput>
```

Figure 5.7: The XML example from the questionnaire, showing two blocks of output expressions for different devices containing one text UI component.

and offers a video, audio and possibly a textual representation (when closed captions are available).

Aside from this, two experts suggested adding semantic annotations at this level to limit which information items can be dropped onto what UI components or devices. This would enable, when the information has been semantically annotated as well, to set up constraints like information items that are semantically tagged as being *food* can only be dropped into UI components that also contain a *food* tag in its attribute. This attribute describes what semantic tags an information item has to possess to be dropped there.

5.5 Interim Conclusion

One strong response was the importance to verify that the interaction concept fits the problem domain and supports the user in the best way possible to achieve her goals. This is only possible when taking the users into consideration from the beginning and by periodically rechecking, for instance via evaluations, that everything is still going in the right direction.

Aside from the strong focus on the problem domain and the users, it has been uncovered that there are many details that contribute (or impair when done wrong) the practicability of a modality-independent interaction concept. The input modalities need to be visible, with the option to deactivate them for certain contexts. Also, interactions should be concise and work seamlessly, avoiding unnecessary steps and reacting in a smart way when the input is incomplete (e.g. guess the best UI component when dropping an item on a device).

6 Summary and Future Work

Concluding this work, a retrospect of what has been discussed is given in this chapter, followed by an outlook on some topics that may be worth pursuing in future work.

As has been elaborated in chapter 3, there are many things to consider when developing a multimodal interaction concept: The definition and separation of the abstract actions from concrete inputs, the design and realisation of UI components that enable and support the interaction concept, concrete mappings that define how to trigger the abstract actions for every supported input modality, identifying and defining meaningful feedback to make the interaction concept visible and predictable in its usage and, finally, how to handle the problems that arise when trying to create a flexible, modality-independent interaction concept.

Another important aspect is the modelling of the information, which needs to match the problem domain to effectively support it. This includes ensuring to have matching content in different representations, adding semantic annotations that play a very important role to support meaningful interactions, supporting hierarchical media to increase the utility by many possibilities and metadata (like dynamic nominators) that can help the user to manage complex information in an easier way in a world that collects and aggregates more and more information every day.

The potential to improve the user experience by utilising multimodal input and output capabilites has been confirmed by the evaluation.

There are still many obstacles to overcome before all of this can be merged into a single, cohesive, seamless user experience, but the foundations for many of these challenges have already been laid out. As this work has proven, it is technically feasible to realise a modality-independent interaction concept that allows the exchange of information across device borders featuring modality-independent representations of the content.

Future Work

The next steps would be to fine-tune the interaction concept and to enhance the prototype based on the suggestions made by the experts. Afterwards, an extensive usability study with end users should be conducted to gain feedback from people who are not as proficient with computer systems as the experts are.

Another important step will be to fully integrate the interaction concept into the existing system. The dialog management, fission and fusion have to be extended to handle the Pick-and-Drop messages received via the message-oriented middleware. Also, these components need to take the additional Pick-and-Drop tags and attributes in the dialog and interaction inputs into consideration. The fission also needs to be changed to take the Pick-and-Drop attributes into account and add a fitting clipboard representation where defined.

Adding semantics to the information storage and the UI is another point to consider. This would add benefits concerning the usability and would allow designing the user interfaces and interactions in a more sensible way, fitting to the problem domain and use cases.

Decoupling the input modalities and their mappings even more by utilising a model-based approach as suggested by an expert in the evaluation is also worth investigating. The modality-independence for input concepts is one of the corner points of this interaction concept and decoupling it into a model-based approach could greatly increase the flexibility and extensibility for input concepts.

Lastly, adding more input modalities to the prototype and conducting an evaluation with a prototype system that supports more input modalities and devices would also provide more insight and concrete evidence on the statements and feedback the experts provided.

A Class Diagrams

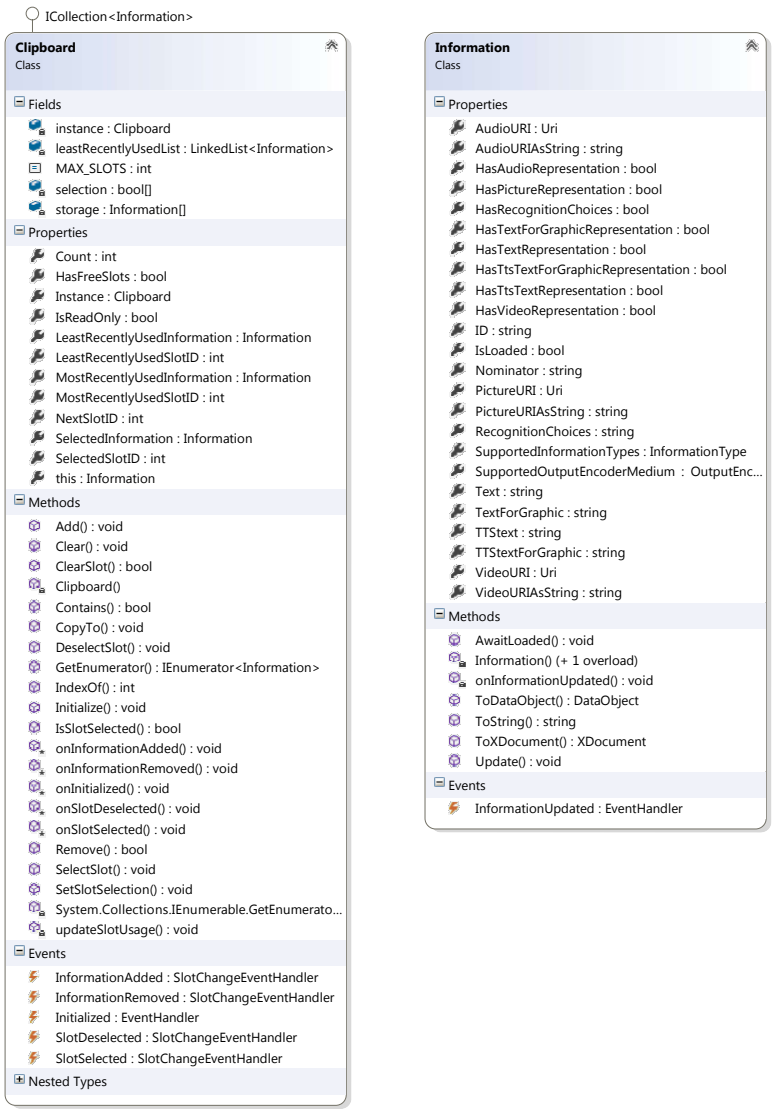

Figure A.1: Full class diagram of the Clipboard and Information classes.

Figure A.2: Full class diagram of the Synchronization namespace.

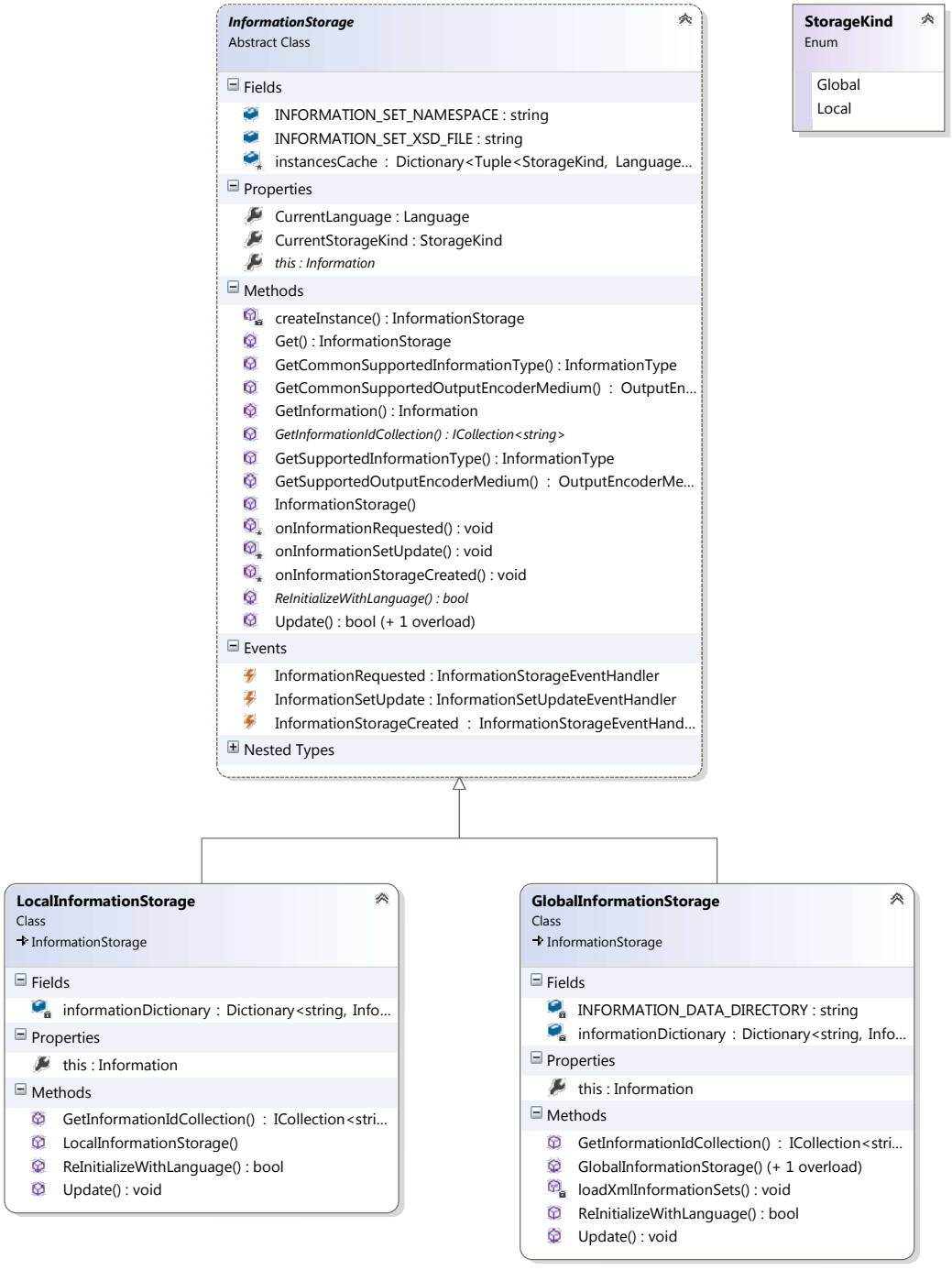

Figure A.3: Full class diagram of the Storage namespace.

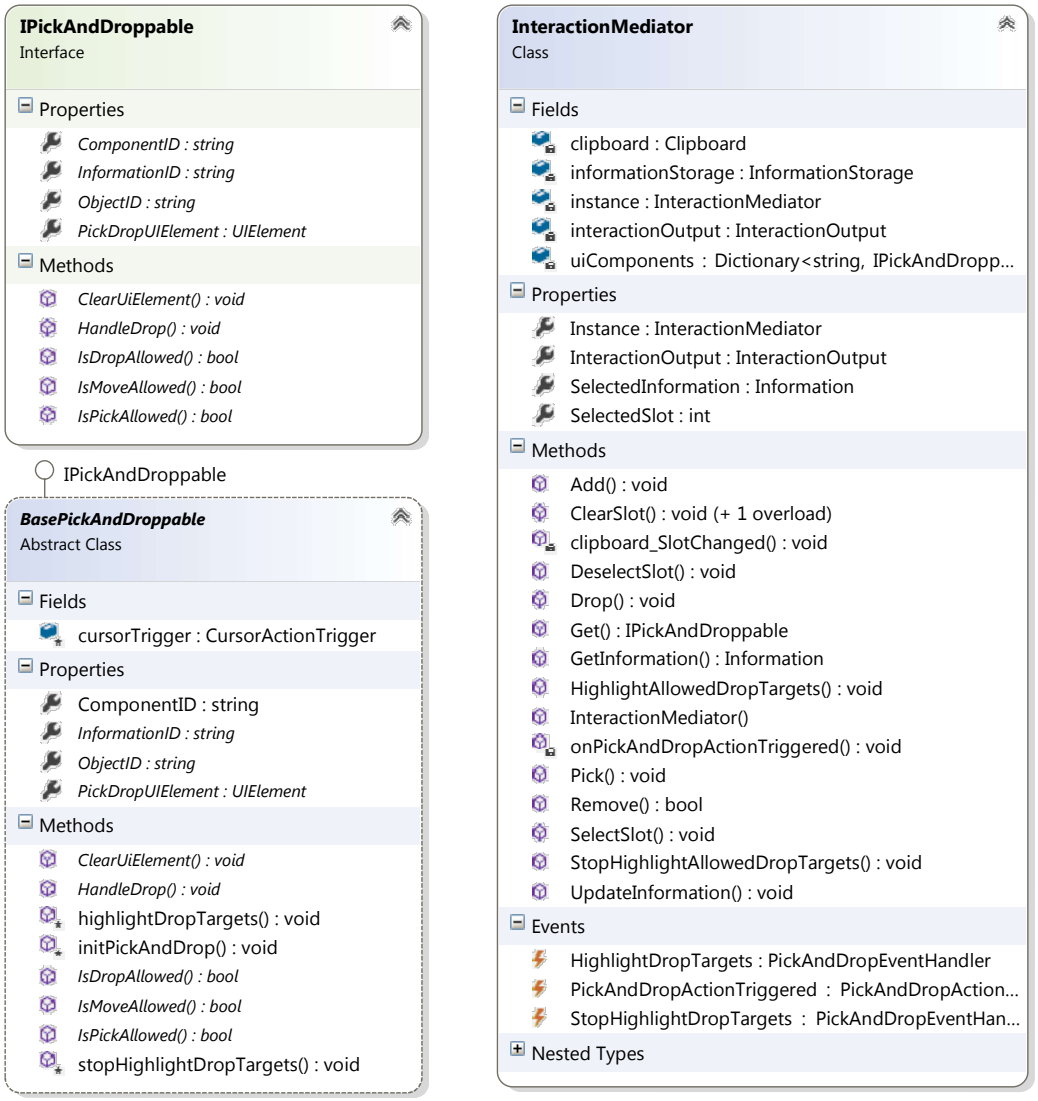

Figure A.4: Full class diagram of the Interaction namespace.

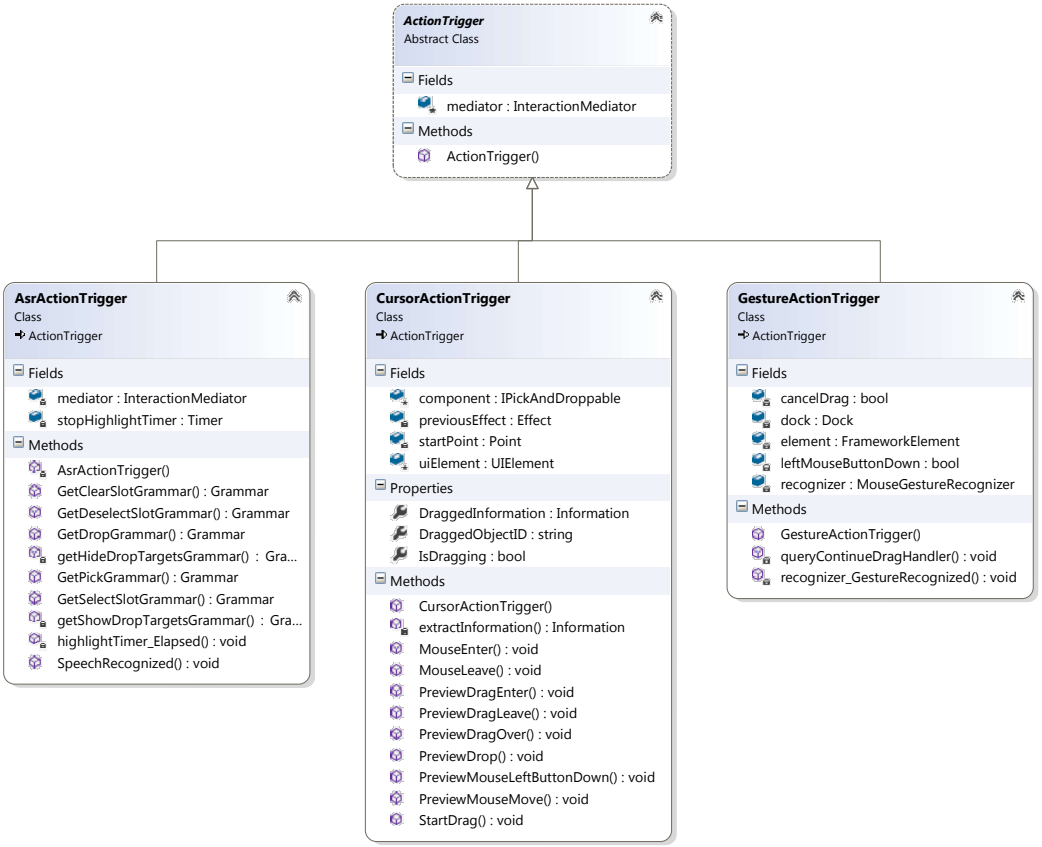

Figure A.5: Full class diagram of the Trigger namespace.

B Tables

GUI	MUI
Single input stream	Multiple input streams
Atomic, deterministic	Continuous, probabilistic
Sequential processing	Parallel processing
Centralised architectures	Distributed & time-sensitive architectures

Table B.1: The differences betweeen GUIs and MUIs in juxtaposition taken from [DLO09].

Element	Description
`Cursor`	The term cursor is used to denote the location designator for triggered actions. It can be either a graphical cursor visible on the screen, like a mouse cursor, or an invisible cursor, like when using a finger to point at something.
`Pickable`	Marks an abstract or concrete component as valid target for `pick` actions.
`Droppable`	Marks an abstract or concrete component as valid target for a `drop` action.
`Information item`	An information item is anything that can be picked up and dropped using the Metamorph interaction concept. It can range from a simple text to an image to a video, to name a few examples. An information item is pickable.
`Slot`	A slot represents a storage unit for a single information item. A slot is droppable.
`Metamorph clipboard`	The Metamorph clipboard is the globally available storage for the picked up information items, offering slots in which information items can be stored. Although the storage itself is invisible, it is made visible through audio feedback and GUI components that are described later.

Table B.2: Overview of the basic elements and terms that are used to define the Metamorph interaction concept.

Action	Description
`Pick`	Store the information item picked up from ObjectID in the clipboard slot with SlotID, the next slot that is free or the slot that was least recently used. **Required data:** `<DeviceID, ComponentID, ObjectID, InformationID, [SlotID]>`
`Drop`	Drop the dragged information item into the designated UI object with ObjectID, initiating a transfer of the information to a droppable UI object. **Required data:** `<DeviceID, ComponentID, ObjectID, InformationID>`
`Select`	Mark the information item stored in the clipboard slot identified by SlotID as selected. **Required data:** `<SlotID>`
`Deselect`	Unmark the information item stored in the clipboard identified by SlotID as selected. **Required data:** `<SlotID>`
`ClearSlot`	Clears the slot identified by SlotID by removing the information item it contains from the clipboard. If no SlotID is given, the item in the selected slot will be removed by default. **Required data:** `<SlotID>`
`ClearUiComponent`	Clear the source UI component with ObjectID, making the drop a cut instead of copy. **Required data:** `<DeviceID, ComponentID, ObjectID>`

Table B.3: Overview of the basic actions defining the Metamorph interaction concept.

Input method	Steps to trigger 'pick'
Mouse (GUI)	1. Click on the desired item with left mouse button 2. Hold the click and start dragging the item i. The Metamorph ring appears 3. Release the click over a slot
Mouse (gesture)	1. Click on the desired item with left mouse button 2. Hold the click and start dragging the item 3. Perform a swipe gesture to the top i. The item will appear in a slot in the Metamorph dock
Keyboard	1. Select the desired item 2. Press Control (ctrl) + x for cut or ctrl + c for copy i. The item will appear in a slot in the Metamorph dock
Touch (GUI)	1. Long touch on the desired item i. The item will be visually 'picked up' i. The Metamorph ring appears 2. Release the touch over a slot
Touch (gesture)	1. Select the desired item i. Item will be visually selected 2. Perform a swipe gesture to the top i. The item will appear in a slot in the Metamorph dock
Android (GUI)	1. Long touch on the desired item i. The item will be visually 'picked up' ii. The Metamorph app appears after a short delay without moving 3. Release the touch over a slot i. The Metamorph app disappears and the last visible app is shown again
Android (gesture)	1. Select the desired item i. Item will be visually selected 2. Perform a swipe gesture to the top i. A notification confirms picking up the item
Voice	1. "Pick «`selection`» [up]", "Take «`selection`»", "Collect «`selection`»" i. Some auditive or visual feedback confirms the pick-up
Body gesture	1. Point on item i. Item will be visually selected 2. Perform a swipe gesture to the top i. The item will appear in a slot in the Metamorph dock

Table B.4: A detailed description of all the actions needed to perform the pick action using all input methods.

Input method	Steps to trigger 'drop'
Mouse (GUI)	1. Click with the left mouse button on the desired item 2. Hold the click and start dragging the item 3. Release the click over the desired location to drop the item
Mouse (gesture)	1. Click and hold the left mouse button 2. Perform a swipe gesture to the bottom i. The item in the selected slot appears at the cursor 3. Release the click over the desired target location to drop the item i. Perform shaking gesture to cancel the drag
Keyboard	1. Press ctrl + v i. The item in the selected slot will be dropped at the cursor's current location
Touch (GUI)	1. Long touch on the desired item i. The item will be visually 'picked up' 2. Release the touch over the desired target location to drop the item
Touch (gesture)	1. Perform a swipe gesture to the bottom i. The item in the selected slot appears at the finger's position 2. Release the touch over the desired target location to drop the item i. Perform shaking gesture to cancel the drag
Android (GUI)	1. Open the Metamorph App 2. Select the desired item 3. Long touch on the desired item i. The item will be visually 'picked up' ii. The Metamorph app disappears and the last visible app is shown again 4. Release the touch over the desired target location to drop the item i. Perform shaking gesture to cancel the drag
Android (gesture)	1. Perform a swipe gesture to the bottom i. The item in the selected slot appears at the finger's position 2. Release the touch over the desired target location to drop the item i. Perform shaking gesture to cancel the drag
Voice	1. "Drop [selection] at «ObjectID»", "Put [selection] into «ObjectID»"
Body gesture	1. Point at the desired target location i. The cursor appears at the desired target location 1. Perform a swipe gesture to the bottom i. The selected items are dropped at the cursors' location

Table B.5: A detailed description of all the actions needed to perform the drop action using all input methods.

Input method	Steps to trigger 'select' and 'deselect'
Mouse (GUI)	1. Click with the left mouse button on an item i. If the item is deselected it will be selected ii. If the item is selected it will be deselected *Alternative* 1. Rightclick and select 'Metamorph ring' from the context menu i. The Metamorph ring appears 2. Click to select or deselect as described before
Mouse (gesture)	1. Perform a circle gesture i. The Metamorph ring appears 2. Click to select or deselect as described under 'Mouse (GUI)'
Keyboard	1. Press Alternative (alt) + Tab until the dock is focussed 2. Use the arrow keys to focus the desired item 3. Press spacebar or enter i. If the item is deselected it will be selected ii. If the item is selected it will be deselected *Alternative* 1. Press ctrl + r i. The Metamorph ring appears 2. Select or deselect using the keyboard as described before
Touch (GUI)	1. Touch on a deselected item i. The item becomes selected 2. Touch on a selected item i. The item becomes deselected
Touch (gesture)	1. Perform a circle gesture i. The Metamorph ring appears 2. Click to select or deselect as described under 'Touch (GUI)'
Android (GUI)	1. Touch on a deselected item i. The item becomes selected 2. Touch on a selected item i. The item becomes deselected
Android (gesture)	3. Perform a circle gesture i. The Metamorph app appears 4. Click to select or deselect as described under 'Android (GUI)'
Voice	"What is in the clipboard?", "What have I picked up?", "Show [me the] contents of the clipboard" *Current Selection* "What [item] is [currently] selected?", "Show|Tell me the selection" *Change selection* "Select|deselect slot «ID»", "Select|deselect «position, like first, second, etc.» item", "Select|deselect item in slot «ID»"
Body gesture	1. Point upwards with the index finger 2. Perform a circle gesture with the fingertip i. The Metamorph ring appears 3. Point for a defined number of seconds on an item to select or deselect the item as described under 'Touch (GUI)'

Table B.6: A detailed description of the actions needed to perform the select or deselect action using all input methods.

Output modality	Feedback	
Visual	**Pick:**	• Item appears at the dock in a slot
	Drag:	• Icon of the picked up item is shown at the cursor
	Drop:	• Item appears at the drop location • A plus icon appears at the cursor when the drop will be a copy
	Item selected:	• Frame and background of the selected slot become hightlighted with a colour
	Item deselected:	• Frame and background of the deselected slots become as usual
	Clipboard contents:	• Metamorph dock (global) • Metamorph ring (local)
	Selected item:	• Frame and background of the selected slot is highlighted with colour
	Representations supported by the items:	• Small general icons below the item's icon that represent text, image, text-to-speech, audio and video representations • Detailed tooltip information on mouse over slot
	Valid drop targets:	• Coloured frame around possible drop targets
Auditive	**Pick:**	• A specific sound with increasing tone height to imply a upward movement
	Drag:	—
	Drop:	• A specific sound with decreasing tone height to imply a downward movement
	Item selected:	• A selection sound
	Item deselected:	• A deselection sound
	Clipboard contents:	• Enumeration of all items by TTS
	Selected item:	• Description of the selected item by TTS
	Representations supported by the items:	• Enumeration of the modalities for a specific given item
	Valid drop targets:	• Enumeration of all possible drop targets' names by TTS

Table B.7: A detailed description of the actions needed to perform the drop action using a set of input concepts.

Bibliography

[AK10] ANUSUYA, M. A. ; KATTI, S. K.: Speech Recognition by Machine, A Review. In: *CoRR* abs/1001.2267 (2010)

[BB00] BARRY BRUMITT, Brian Meyers Steven S. John Krumm K. John Krumm: Ubiquitous computing and the role of geometry. In: *IEEE Personal Communications* 7 (2000), S. 41–43

[Bol80] BOLT, Richard A.: "Put-that-there": Voice and gesture at the graphics interface. In: *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1980 (SIGGRAPH '80). – ISBN 0–89791–021–4, 262–270

[BP04] BANDELLONI, Renata ; PATERNÒ, Fabio: Migratory user interfaces able to adapt to various interaction platforms. In: *Int. J. Hum.-Comput. Stud.* 60 (2004), Nr. 5-6, 621-639. http://dblp.uni-trier.de/db/journals/ijmms/ijmms60.html#BandelloniP04

[BP05] BERTI, Silvia ; PATERNÒ, Fabio: Migratory MultiModal interfaces in MultiDevice environments. In: *Proceedings of the 7th international conference on Multimodal interfaces*. New York, NY, USA : ACM, 2005 (ICMI '05). – ISBN 1–59593–028–0, 92–99

[CWC05] CLERCKX, Tim ; WINTERS, Frederik ; CONINX, Karin: Tool support for designing context-sensitive user interfaces using a model-based approach. In: *Proceedings of the 4th international workshop on Task models and diagrams*. New York, NY, USA : ACM, 2005 (TAMODIA '05). – ISBN 1–59593–220–8, 11–18

[DLO09] DUMAS, Bruno ; LALANNE, Denis ; OVIATT, Sharon: Multimodal Interfaces: A Survey of Principles, Models and Frameworks. Version: 2009. http://dx.doi.org/10.1007/978-3-642-00437-7_1. In: LALANNE, Denis (Hrsg.) ; KOHLAS, Jörg (Hrsg.): *Human Machine Interaction*. Berlin, Heidelberg :

Springer-Verlag, 2009. – ISBN 978–3–642–00436–0, Kapitel Multimodal Interfaces: A Survey of Principles, Models and Frameworks, 3-26

[FWW11] FINDLATER, Leah ; WOBBROCK, Jacob O. ; WIGDOR, Daniel: Typing on flat glass: examining ten-finger expert typing patterns on touch surfaces. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2011 (CHI '11). – ISBN 978–1–4503–0228–9, 2453–2462

[HDS12] HOSTE, Lode ; DUMAS, Bruno ; SIGNER, Beat: SpeeG: a multimodal speech- and gesture-based text input solution. In: *Proceedings of the International Working Conference on Advanced Visual Interfaces*. New York, NY, USA : ACM, 2012 (AVI '12). – ISBN 978–1–4503–1287–5, 156–163

[HSW12] HONOLD, Frank ; SCHÜSSEL, Felix ; WEBER, Michael: Adaptive probabilistic fission for multimodal systems. In: *Proceedings of the 24th Australian Computer-Human Interaction Conference*. New York, NY, USA : ACM, 2012 (OzCHI '12). – ISBN 978–1–4503–1438–1, 222–231

[HSW+13] HONOLD, Frank ; SCHÜSSEL, Felix ; WEBER, Michael ; NOTHDURFT, Florian ; BERTRAND, Gregor ; MINKER, Wolfgang: Context Models for Adaptive Dialogs and Multimodal Interaction, 2013 (Intelligent Environments)

[Kab10] KABRT, Lukas: *Mouse Gestures for .NET*. As seen on: 22.07.2013. `http://mousegestures.codeplex.com/`. Version: March 2010

[KI07] KOBAYASHI, Masatomo ; IGARASHI, Takeo: Boomerang: Suspendable drag-and-drop interactions based on a throw-and-catch metaphor. In: *Proceedings of the 20th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 2007 (UIST '07). – ISBN 978–1–59593–679–0, 187–190

[Mar99] MARTIN, J.C.: TYCOON: six primitive types of cooperation for observing, evaluating and specifying cooperations. In: *AAAI Technical Report FS-99-03* (1999)

[Med07] MEDERO, Shawn: *Paper Prototyping*. alistapart.com, as seen on: 20.05.2013. `http://alistapart.com/article/paperprototyping`. Version: January 2007

[Mil56] MILLER, George: *The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*. `http://cogprints.org/730/`.

Version: 1956. – One of the 100 most influential papers in cognitive science: http://cogsci.umn.edu/millennium/final.html

[MM99] MILLER, Robert C. ; MYERS, Brad A.: Synchronizing clipboards of multiple computers. In: *Proceedings of the 12th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 1999 (UIST '99). – ISBN 1–58113–075–9, 65–66

[MSB91] MACKENZIE, I. S. ; SELLEN, Abigail ; BUXTON, William A. S.: A comparison of input devices in element pointing and dragging tasks. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 1991 (CHI '91). – ISBN 0–89791–383–3, 161–166

[Mye98] MYERS, Brad A.: A brief history of human-computer interaction technology. In: *interactions* 5 (1998), März, Nr. 2, 44–54. http://dx.doi.org/10.1145/274430.274436. – DOI 10.1145/274430.274436. – ISSN 1072–5520

[Nie00] NIELSEN, Jakob: *Why You Only Need to Test with 5 Users*. As seen on: 21.08.2013. http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. Version: March 2000

[Nor02] NORMAN, Donald A.: *The Design of Everyday Things*. Reprint Paperback. New York : Basic Books, 2002. – ISBN 0–465–06710–7

[Ovi03] OVIATT, Sharon: The human-computer interaction handbook. Version: 2003. http://dl.acm.org/citation.cfm?id=772072.772093. In: JACKO, Julie A. (Hrsg.) ; SEARS, Andrew (Hrsg.): *The human-computer interaction handbook*. Hillsdale, NJ, USA : L. Erlbaum Associates Inc., 2003. – ISBN 0–8058–3838–4, Kapitel Multimodal interfaces, 286–304

[Pap13] *Paper prototyping*. Wikipedia, as seen on: 20.05.2013. http://en.wikipedia.org/wiki/Paper_prototyping. Version: March 2013

[Pet10] PETRASCH, Roland: Model based user interface development with HCI patterns: variatio delectat. In: *Proceedings of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems*. New York, NY, USA : ACM, 2010 (PEICS '10). – ISBN 978–1–4503–0246–3, 10–11

[PKH08] PARK, In-Kwon ; KIM, Jung-Hyun ; HONG, Kwang-Seok: An implementation of an FPGA-based embedded gesture recognizer using a data glove. In: *Proceedings of the 2nd international conference on Ubiquitous information management*

and communication. New York, NY, USA : ACM, 2008 (ICUIMC '08). – ISBN 978–1–59593–993–7, 496–500

[Ram03] RAMAN, T. V.: *User Interface Principles For Multimodal Interaction*. http://www.cim.mcgill.ca/~jer/courses/hci/ref/mmi-position.html. Version: 2003

[Rek97] REKIMOTO, Jun: Pick-and-drop: A direct manipulation technique for multiple computer environments. In: *Proceedings of the 10th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 1997 (UIST '97). – ISBN 0–89791–881–9, 31–39

[SDD12] SONG, Yale ; DEMIRDJIAN, David ; DAVIS, Randall: Continuous body and hand gesture recognition for natural human-computer interaction. New York, NY, USA : ACM, März 2012. – ISSN 2160–6455, 5:1–5:28

[Seb09] SEBE, Nicu: Multimodal interfaces: Challenges and perspectives. In: *J. Ambient Intell. Smart Environ.* 1 (2009), Januar, Nr. 1, 23–30. http://dl.acm.org/citation.cfm?id=1735821.1735824. – ISSN 1876–1364

[SEM] *SEMAINE Trac system*. As seen on: 19.07.2013. http://semaine.opendfki.de/

[SER09] STOLEE, Kathryn T. ; ELBAUM, Sebastian ; ROTHERMEL, Gregg: Revealing the copy and paste habits of end users. In: *Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Washington, DC, USA : IEEE Computer Society, 2009 (VLHCC '09). – ISBN 978–1–4244–4876–0, 59–66

[SGP00] SILVA, Paulo Pinheiro d. ; GRIFFITHS, Tony ; PATON, Norman W.: Generating user interface code in a model based user interface development environment. In: *Proceedings of the working conference on Advanced visual interfaces*. New York, NY, USA : ACM, 2000 (AVI '00). – ISBN 1–58113–252–2, 155–160

[SSR13] SEIFERT, Julian ; SCHNEIDER, Dennis ; RUKZIO, Enrico: Extending Mobile Interfaces with External Screens. In: *In Proc. of Interact 2013 (IFIP TC13 Conference on Human-Computer Interaction), Springer, 8 pages.*, 2013

[SSRG12] SCHMIDT, Dominik ; SEIFERT, Julian ; RUKZIO, Enrico ; GELLERSEN, Hans: A cross-device interaction style for mobiles and surfaces. In: *Proceedings of the*

Designing Interactive Systems Conference. New York, NY, USA : ACM, 2012 (DIS '12). – ISBN 978–1–4503–1210–3, 318–327

[Twa] http://www.addictedtocoffee.de

[VC04] VANDERVELPEN, Chris ; CONINX, Karin: Towards model-based design support for distributed user interfaces. In: *Proceedings of the third Nordic conference on Human-computer interaction*. New York, NY, USA : ACM, 2004 (NordiCHI '04). – ISBN 1–58113–857–1, 61–70

[Vel09] VELLIS, George: Model-based development of synchronous collaborative user interfaces. In: *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. New York, NY, USA : ACM, 2009 (EICS '09). – ISBN 978–1–60558–600–7, 309–312

[VLM+04] VANDERDONCKT, Jean ; LIMBOURG, Quentin ; MICHOTTE, Benjamin ; BOUILLON, Laurent ; TREVISAN, Daniela ; FLORINS, Murielle: UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In: *Proceedings of W3C Workshop on Multimodal Interaction WMI'2004*. Sophia Antipolis, Juli 2004, S. 35–42

[WCO95] WAGNER, Annette ; CURRAN, Patrick ; O'BRIEN, Robert: Drag me, drop me, treat me like an object. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1995 (CHI '95). – ISBN 0–201–84705–1, 525–530

[Wei95] WEISER, Mark: The computer for the 21st century. Version: 1995. http://dl.acm.org/citation.cfm?id=212925.213017. In: BAECKER, Ronald M. (Hrsg.) ; GRUDIN, Jonathan (Hrsg.) ; BUXTON, William A. S. (Hrsg.) ; GREENBERG, Saul (Hrsg.): *Human-computer interaction*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1995. – ISBN 1–55860–246–1, Kapitel The computer for the 21st century, 933–940

[Wik13a] *Clipboard (computing)*. Wikipedia, as seen on: 3.09.2013. http://en.wikipedia.org/wiki/Clipboard_(computing). Version: July 2013

[Wik13b] *Cut, copy, and paste*. Wikipedia, as seen on: 3.06.2013. http://en.wikipedia.org/wiki/Copy_and_paste. Version: August 2013

[Wik13c] *Drag and drop*. Wikipedia, as seen on: 3.09.2013. http://en.wikipedia.org/wiki/Drag_and_drop. Version: August 2013

[Wik13d] *Media (communication)*. Wikipedia, as seen on: 3.06.2013. http://en.wikipedia.org/wiki/Media_(communication). Version: May 2013

[Wik13e] *Modality (human-computer interaction)*. Wikipedia, as seen on: 26.05.2013. http://en.wikipedia.org/wiki/Modality_(human%E2%80%93computer_interaction). Version: April 2013

[Wik13f] *Siri*. Wikipedia, as seen on: 2.09.2013. http://en.wikipedia.org/wiki/Siri. Version: September 2013

[WLD08] WITT, Hendrik ; LAWO, Michael ; DRUGGE, Mikael: Visual feedback and different frames of reference: the impact on gesture interaction techniques for wearable computing. In: *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*. New York, NY, USA : ACM, 2008 (MobileHCI '08). – ISBN 978–1–59593–952–4, 293–300

[ZNK10] ZHANG, Rui ; NORTH, Stephen ; KOUTSOFIOS, Eleftherios: A comparison of speech and GUI input for navigation in complex visualizations on mobile devices. In: *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*. New York, NY, USA : ACM, 2010 (MobileHCI '10). – ISBN 978–1–60558–835–3, 357–360

Name: Michael Barth Matrikelnummer: 748890

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Michael Barth

Picture Credits

The file icons used in figure 3.5 are freeware and were created by Oliver Twardowski [Twa].

All images used in the concept art and screenshots of the Metamorph app, dock and ring (see figures 3.9, 3.7 and 3.8, amongst others) were found using the Google Image Search and are the property of their respective owners.

The green apple was taken from: `http://iran-banner.com/`

The keikogi was taken from: `http://www.budovideos.com/`

The Spaceballs cover was taken from: `http://covers.box.sk/`
Spaceballs: The Movie is copyrighted by *Metro-Goldwyn-Mayer Studios Inc.*, all rights reserved.

The Uni Ulm logo was taken from: `http://www.uni-ulm.de/`

The icons used in the dock prototype (see figure 4.13) were found on the website of `http://www.findicons.com/` and are property of their respective owners.

The red trash bin was created by: `http://bombiadesign.com/`