Maastricht University | universiteit hasselt
KNOWLEDGE IN ACTION

2014 | Faculty of Sciences

DOCTORAL DISSERTATION

Designing for Intelligibility and Control in Ubiquitous Computing Environments

Doctoral dissertation submitted to obtain the degree of
Doctor of Science: Information Technology, to be defended by

Jo Vermeulen

Promoter: Prof. Dr Karin Coninx | UHasselt / tUL
Co-promoter: Prof. Dr Kris Luyten | UHasselt / tUL

ABSTRACT

In this dissertation, I explore designing for *intelligibility* and *control* in ubiquitous computing applications, in particular, in context-aware systems. Earlier work has identified a number of interaction challenges that users face when dealing with context-aware systems, such as being unable to understand why the system is responding in a certain way, or being unable to intervene when the system makes a mistake. These challenges impact users' feelings of trust and of being in control, and could eventually lead to users disengaging from interaction with the system altogether. It is important to address these interaction challenges to allow for smooth integration of ubicomp technologies into our lives. For this purpose, two general principles have been proposed that should be supported by context-aware systems: *intelligibility* and *control*. Intelligibility is the ability of a context-aware system to present itself and its behaviour to its users, while control deals with allowing users to step in and *intervene* to correct the system. Although a number of techniques have been explored to improve intelligibility and control for context-aware systems, it is not yet clear how ubiquitous computing researchers, developers and interaction designers should design for intelligibility and control.

This dissertation serves as a design space exploration to inform the design of future ubiquitous computing applications that provide support for intelligibility and control. I aim for this work to be both *generative* by guiding designers in exploring various ways to support intelligibility and control, and *generalizable* by exploring techniques that can be applied by interaction designers in a wide range of ubiquitous computing scenarios. In particular, I provide the following three major contributions:

First, I present a design space that captures different decisions that designers face when adding support for intelligibility and control. This design space consists of six dimensions and can be used both as an analytical tool to classify and compare different techniques, and can help designers explore alternative designs.

Second, I present general design principles and techniques that can be applied in a wide range of ubicomp scenarios. I contribute three general techniques that can be used at three different times during the interaction: *feedforward* (before actions), *slow-motion feedback* (during actions), and *why questions* (after actions).

Third, I describe an in-depth case study of supporting intelligibility and control in proxemic interactions. This can serve as inspiration for designers and researchers looking to consider these principles in different ubicomp applications. In particular, I propose the use of a secondary, assisting floor display that informs users about the tracking status, indicates action possibilities, and invites and guides users throughout their interaction with the primary display.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the support and encouragement of several people.

First, I would like to thank my advisor Prof. Dr Karin Coninx. Karin, thank you for providing me with the opportunity to pursue a PhD, for your support and encouragement throughout the years, and for giving me the freedom to pursue my own research interests. I truly enjoyed the many discussions we had during the course of writing this dissertation. Thank you for the countless suggestions and comments that helped shape this work into its current form. I would also like to thank my co-advisor Prof. Dr Kris Luyten, without whom I might not have embarked on this journey in the first place. Kris, thank you for sparking my interest in HCI research as an undergraduate student, and for your endless encouragement, energy and enthusiasm. Your feedback and suggestions had a large impact on the ideas presented here. During the final two years of my PhD, I also had the opportunity to work with Prof. Dr Johannes Schöning. Johannes, thank you for your invaluable advice (usually provided over a morning coffee), and for teaching me about German efficiency.

I would like to thank the members of my jury: Prof. Dr Wim Lamotte, Prof. Dr Johannes Schöning, Em. Prof. Dr Joëlle Coutaz, Prof. Dr Hans Gellersen, and Dr Nicolai Marquardt: thank you for your constructive and valuable feedback that helped to improve this dissertation. I also like to thank the chairmain of my jury, Prof. Dr Marc Gyssens.

I would like to express my gratitude to the EDM management, Prof. Dr Eddy Flerackers and Prof. Dr Frank Van Reeth, for providing me with the opportunity to work in such a supportive environment. I also thank Ingrid Konings and Roger Claes for helping me with administrative tasks and taking care of logistics, and Luc Adriaens for audiovisual support.

Thank you to all my friends and colleagues in the EDM HCI group. Special thanks to Dr Davy Vanacken for his helpful advice during my first years as a teaching assistant. I would also like to single out Dr Lode Vanacken and Dr Jan Meskens for all the interesting research discussions we had during the course of my PhD. Thank you to Dr Geert Vanderhulst for being an amazing collaborator on PervasiveCrystal. I also like to thank Daniël Teunkens for sketching the scenarios used for PervasiveCrystal, and Karel Robert for creating the visual designs used for the Visible Computer. Thanks to Dr Mieke Haesen for acting so convincingly in the Visible Computer video, and to Raf Ramakers for helping me shoot the ProxemicFlow video. Also thanks to all my other colleagues in the EDM HCI group that contributed to making my time there a fantastic experience: Kashyap Todi, Dr Tim Clerckx, Dr Yves Vandriessche, Dr Sean Tan, Tom De Weyer, Gustavo Rovelo Ruiz, Kris Gabriëls, Kristof Thys, Dr Petr Aksenov, Sofie Notelaers, Dr Jan Van den Bergh, Donald Degraen, Dr Alexandre Demeure, Pavel Samsonov and many others. Also thank you

to the many amazing master's students who I co-supervised, with special thanks to Gert Vos and Jonathan Slenders who contributed to the work presented in this dissertation.

I also had the opportunity to work with several external collaborators during the course of my PhD. I would like to especially thank Dr Nicolai Marquardt for our wonderful collaboration, for his support and for all the things I learned while working on our projects in London. I also want to particularly thank Prof. Dr Hans Gellersen for providing me with the opportunity to spend a couple of weeks at his research group at Lancaster University. Thank you to Dr Fahim Kawsar and Prof. Dr Gerd Kortuem for our collaboration on the situated glyphs project. Fahim, also thanks for all your advice over the past years. I also thank: Prof. Dr Elise van den Hoven, Prof. Dr Saul Greenberg, Prof. Dr Sebastian Boring, Jakub Dostal, Sarah Mennicken, Prof. Dr Elaine Huang, Victor Cheung, Diane Watson, Prof. Dr Mark Hancock, Prof. Dr Stacey D. Scott, Steven Houben, Prof. Dr Jakob Bardram, Dr Adalberto L. Simeone and Kevin Smith.

I met some amazing PhD students along the way, and I particularly want to thank Sarah Mennicken, Dr Brian Y. Lim, Dr Rémi Barraquand, and Lindsay MacDonald. Special thanks to Lindsay and Sarah for proof-reading early drafts of this dissertation.

I would like to thank my family and friends for their support. In particular, thank you to my parents for their encouragement, and for taking care of so many little things that made my life easier in the last few months. I would especially like to thank my father for sparking my interest in science, and always encouraging me to challenge myself and explore the limits of my abilities. Special thanks to my late uncle Tony who unfortunately passed away while I was pursuing this degree. Thank you for your feedback on my early papers, which greatly improved my English writing skills. I wish I could have shared this accomplishment with you. Also thanks to my aunt Lieve for all her support and encouragement, and to my cousin Sarah for doing amazing voice-overs for my research videos. I also like to thank my friends for the necessary diversions from work. In particular, thank you to Bart Dehairs, Davy Jooken, Johan Nulens, Dr Lode Vanacken and Luk Vloemans for always being available for support and distraction.

PUBLICATIONS

Materials, ideas and figures have appeared previously in the following publications. A full list of publications is available in Appendix A.

CONFERENCE PAPERS

- **Jo Vermeulen**, Kris Luyten, Karin Coninx, and Nicolai Marquardt. The Design of Slow-Motion Feedback. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 267–270, 2014. ACM. ISBN 978-1-4503-2902-6.
- **Jo Vermeulen**, Kris Luyten, Elise van den Hoven, and Karin Coninx. Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1931–1940, 2013. ACM. ISBN 978-1-4503-1899-0.
- **Jo Vermeulen**, Kris Luyten and Karin Coninx. Intelligibility Required: How to Make us Look Smart Again. In *Proceedings of the 10th Romanian Conference on Human-Computer Interaction*, ROCHI '13, 2013.
- **Jo Vermeulen**, Kris Luyten, and Karin Coninx. Understanding Complex Environments with the Feedforward Torch. In *Ambient Intelligence*, volume 7683 of *Lecture Notes in Computer Science*, pages 312–319. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34897-6.
- **Jo Vermeulen**, Fahim Kawsar, Adalberto L. Simeone, Gerd Kortuem, Kris Luyten, and Karin Coninx. Informing the Design of Situated Glyphs for a Care Facility. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, VLHCC '12, pages 89–96, 2012.
- Fahim Kawsar, **Jo Vermeulen**, Kevin Smith, Kris Luyten, and Gerd Kortuem. Exploring the Design Space for Situated Glyphs to Support Dynamic Work Environments. In *Pervasive Computing*, volume 6696 of *Lecture Notes in Computer Science*, pages 70–78. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21725-8.
- **Jo Vermeulen**, Geert Vanderhulst, Kris Luyten, and Karin Coninx. PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications. In *Intelligent Environments (IE), 2010 Sixth International Conference on*, pages 271–276, 2010. IEEE.
- **Jo Vermeulen**, Jonathan Slenders, Kris Luyten, and Karin Coninx. I Bet You Look Good on the Wall: Making the Invisible Computer Visible. In *Ambient*

Intelligence, volume 5859 of *Lecture Notes in Computer Science*, pages 196–205. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-05407-5.

EXTENDED ABSTRACTS

- **Jo Vermeulen**. Improving Intelligibility and Control in Ubicomp. In *Adjunct Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, Ubicomp '10 Adjunct, pages 485–488, 2010. ACM. ISBN 978-1-4503-0283-8.
- **Jo Vermeulen**, Geert Vanderhulst, Kris Luyten, and Karin Coninx. Answering Why and Why Not Questions in Ubiquitous Computing. In *Proceedings of the 11th ACM International Conference on Ubiquitous Computing – Adjunct Papers*, Ubicomp '09 Adjunct, pages 210–213, 2009.

RESEARCH COLLABORATION ACKNOWLEDGEMENTS

The research presented in this dissertation was not undertaken by me alone. Even though a dissertation is mostly an individual contribution to science, it would not be possible without successful and inspiring collaborations with other researchers. Here, I acknowledge the contributions of a talented group of research collaborators both at Hasselt University and at external institutions. Without their efforts, this research could not have been realized in its current scope. My first and foremost collaborators were my advisor Prof. Dr Karin Coninx and co-advisor Prof. Dr Kris Luyten, under whose supervision I undertook this dissertation research.

CHAPTER 4 The situated glyphs prototype and system were developed by a team at Lancaster University (UK), led by Prof. Dr Gerd Kortuem and Dr Fahim Kawsar. I contributed to the conceptual design of situated glyphs, and designed and conducted the study at Mainkofen District Hospital in collaboration with Dr Fahim Kawsar, Prof. Dr Gerd Kortuem and Esunly Medina. This research resulted in two publications: one paper detailing the situated glyphs design space (Kawsar et al., 2011) and a follow-up study (Vermeulen et al., 2012a) at another care facility. Co-authors on these publications are Fahim Kawsar, Gerd Kortuem, Kris Luyten, Karin Coninx, Adalberto L. Simeone and Kevin Smith.

CHAPTER 5 The research on reframing feedforward (Vermeulen et al., 2013b) was done in collaboration with Prof. Dr Elise van den Hoven (TU Eindhoven, Netherlands). Gert Vos implemented the Feedforward Torch prototype and conducted the study for his Master's thesis at Hasselt University, under the supervision of me and my advisors. I did the conceptual work on the Feedforward Torch and iteration on prototyping, and wrote the resulting publication (Vermeulen et al., 2012b).

CHAPTER 6 The conceptualization of slow-motion feedback (Vermeulen et al., 2014) was done in collaboration with Dr Nicolai Marquardt (University College London, UK). The research on The Visible Computer (Vermeulen et al., 2009a) was conducted in collaboration with Jonathan Slenders during his research internship at Hasselt University, under supervision of me and my advisors. Jonathan Slenders implemented the prototype of the Visible Computer; the conceptual foundation for the prototype and iteration on visualization designs were done by me, in addition to the design and execution of the study.

CHAPTER 7 The research on PervasiveCrystal (Vermeulen et al., 2010) was conducted in collaboration with Dr Geert Vanderhulst at Hasselt University. Part of the implementation was done collaboratively with Dr Geert Vanderhulst, but the majority of programming work was done by me. The study was designed and conducted by me.

CHAPTER 8 The research on Proxemic Flow was done in collaboration with Dr Nicolai Marquardt (University College London, UK), Dr Jon Bird (City University London, UK) and my advisors. The conceptual work for the floor visualizations was done in collaboration with Dr Nicolai Marquardt. Dr Jon Bird developed the LED floor display hardware and initial software. I led the design and implementation of the Proxemic Flow architecture, including the floor toolkit, tracking and rendering components, and the alternative projector-based setup. The majority of the programming work was done by myself, but parts with respect to integrating the floor display hardware and software were done collaboratively with Dr Nicolai Marquardt. A paper describing Proxemic Flow co-authored by the above team is still in submission at the time of publication of this dissertation.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

1.1 MOTIVATION

In the last decade, we have seen computing being increasingly integrated into our everyday lives. Computers have moved from the desktop into our pockets, onto our wrists and into our homes, where they are being integrated into everyday household appliances such as thermostats and vacuum cleaners. In terms of performance, today's mobile computers are yesterday's supercomputers. They use an ever-growing array of sensors and sophisticated algorithms to automatically respond to the situation in which they are used, making them *context-aware*. In this sense, one could say that we have arrived at Weiser's vision of ubiquitous computing (Weiser, 1991).

The increasing sophistication of computers and their advanced sensing capabilities might indicate that they have become easier to use, but this is not necessarily the case. At the heart of the problem lies the great paradox of context-aware computing: as machines become 'smarter', have more capabilities to sense what we are doing, and attempt to autonomously respond to those changes, they can actually become harder to use. When context-aware systems work well, they correspond to Weiser's ideal of calm computing (Weiser and Brown, 1995) and can indeed make our lives easier. However, when they fail, they become an inconvenience and can make our lives more difficult, requiring us to constantly monitor them (Norman, 2009). In contrast, traditional 'dumb' computers behave in a more expected manner, that is to say, they tend to only act when they are told to.

In "The Design of Future Things", Norman (2009, pg. 13) argues that machines themselves are not intelligent, rather, their intelligence resides in the mind of the designer, who tries to imagine all possible situations that might occur and devise solutions to each of these situations. It is, however, unlikely that the designer can predict every situation that may cause errors in the operation of the machine, and even more unlikely that the designer will be present to devise a solution to unexpected situations. Even systems that rely on sophisticated inferencing algorithms are trained to recognize and respond to specific types of situations. Computers will continue to have difficulties in dealing with unexpected situations, and this issue will persist for many years to come (Rogers, 2006). Moreover, when computers interact with humans, unexpected situations are the norm rather than the exception. Context-aware systems will make mistakes as there are usually unique human aspects of the context that cannot reliably be sensed (Bellotti and Edwards, 2001).

The difficulty of building robust context-aware systems and the interaction challenges users face when interacting with context-aware systems have been identified in previous work (e.g., Bellotti and Edwards, 2001; Bellotti et al., 2002; Cheverst et al., 2001; Erickson, 2002; Greenberg, 2001). In the literature, two important principles

have been proposed to address interaction challenges with context-aware computing: *intelligibility* and *control*. Intelligibility is the ability of a context-aware system to present itself and its behaviour to its users: what it knows, how it knows this, and how it is using that information (Bellotti and Edwards, 2001). Control, on the other hand, deals with allowing users to intervene when the system makes a mistake (e.g., to cancel or control a system action in progress). There are also some situations in which systems require user involvement (e.g., because there is an increased risk of the system making the wrong decision). It is important to note that intelligibility is a prerequisite for control: before users can intervene, systems must provide them with sufficient information about what they have done and are currently doing.

Although a number of techniques have been explored to improve intelligibility and control for context-aware systems, such as providing textual explanations (Lim and Dey, 2010), it is not yet clear how ubiquitous computing (ubicomp) researchers, developers and interaction designers should design for intelligibility and control. The availability of a library of existing examples and a set of reusable design principles that can be employed could lay the groundwork that is required to resolve the difficulties of smoothly integrating ubicomp technologies into our lives.

1.2 APPROACH AND METHODOLOGY

In this dissertation, I approach providing support for intelligibility and control in context-aware systems from a design perspective. This work serves as a *design space exploration* that aims to be:

- *generative* by providing design dimensions that can inform the design and selection of various ways to support intelligibility and control; and
- *generalizable* by presenting design principles and techniques that can be applied in a wide range of ubicomp scenarios.

Similar to generative theories in human-computer interaction (Rogers, 2004; Shneiderman, 2006), which provide design dimensions and constructs to generate new ideas and inform novel designs, this design space exploration can be used to inform and guide ubicomp interaction designers, researchers, and developers in creating innovative design solutions to support intelligibility and control. Based on a review and analysis of the literature, and informed by insights from designing context-aware systems for demanding work environments (Chapter 4), I present several design dimensions that capture different decisions that designers face when adding support for intelligibility and control (Chapter 3). I then analyse different designs as points in a multi-dimensional space formed by the design space dimensions, and contribute a number of novel techniques within this space (see Section 9.2.3 for an overview). This dissertation particularly focuses on the *timing* dimension in the design space, as this dimension has not been extensively explored in the literature before. To further increase the generative power of the presented design space (Beaudouin-Lafon, 2004), I introduce general design principles and techniques along the timing dimension (Chapters 5–7). Finally, I discuss a case study

of supporting intelligibility and control for proxemic interactions that can serve as inspiration for designers looking to apply the work presented in this dissertation to different ubicomp applications (Chapter 8).

Given the multi-dimensional nature of the problem, I use a multi-faceted methodological approach, combining several research methods. As mentioned previously, I performed a design space analysis in Chapter 3 to provide an overview of design decisions related to intelligibility and control. Additionally, I conducted an in-depth literature review to provide a refined definition of feedforward and disambiguate it from perceived affordances and feedback (Chapter 5). The design-centric perspective is combined with technology-centric (i.e., prototypes and technical frameworks) and analysis-centric perspectives (i.e., user studies), where appropriate. In particular, several prototypes were developed to illustrate the proposed design principles, concepts and techniques, and to demonstrate their feasibility (e.g., the Feedforward Torch in Section 5.6, the Visible Computer in Section 6.3, PervasiveCrystal in Chapter 7, and Proxemic Flow in Chapter 8). The resulting proof-of-concept systems provide an underlying platform to further explore different interactions. Additionally, I gathered informal user feedback on several of the presented techniques, which allowed me to refine my ideas, inform future design iterations and reflect on the presented design space.

1.3 CONTRIBUTIONS

This dissertation makes the following major contributions:

1. A *design space for intelligibility and control* (Chapter 3) that captures different decisions that designers face when adding support for intelligibility and control. The design space can be used as an analytical tool and can help designers with exploring alternative designs.

2. An *in-depth exploration of the timing dimension* in this design space. In particular, I contribute three general techniques that can be used at three different times during the interaction: before, during, and after actions.

    a) *Before*—The design principle *feedforward* (Chapter 5), which informs users of the results of their actions. A new definition of feedforward is provided that further disambiguates feedforward from feedback and affordances. Several existing examples of feedforward are discussed, including the *Feedforward Torch* technique. Additionally, I identify four new classes of feedforward: *hidden*, *false*, *sequential*, and *nested* feedforward.

    b) *During*—The design principle *slow-motion feedback* (Chapter 6), aimed at allowing users to intervene during system actions by having the system slow down when taking action and provide intermediate feedback. I illustrate the application of slow-motion feedback in the *Visible Computer* system to provide real-time and in-place feedback in context-aware environments using projected visualizations. Furthermore, I introduce a de-

sign space to reason about when and how feedback is provided, and use this to analyse notable existing applications of slow-motion feedback.

c) *After*—The ability to *pose why questions about the behaviour of a context-aware system* (Chapter 7). This allows users to gain an understanding of how the system works by receiving intelligible explanations of why it acted in a certain way. I introduce *PervasiveCrystal*, a framework for building context-aware applications that can provide answers to 'why?' and 'why not?' questions about their behaviour.

3. A *case study* (Chapter 8) in supporting intelligibility and control for *proxemic interactions*, a subdomain of context-aware computing. I discuss the design and implementation of dynamic peripheral floor visualizations to address interaction challenges with proxemic-aware interactive surfaces. The *Proxemic Flow* system uses a floor display that plays a secondary, assisting role to aid users in interacting with the primary display. The floor informs users about the tracking status, indicates action possibilities, and invites and guides users throughout their interaction with the primary display.

1.4 DISSERTATION OUTLINE

This dissertation consists of nine chapters including this introductory chapter and a concluding chapter. As a guide to the organization of the remainder of this dissertation, I provide a brief overview of the following chapters:

CHAPTER 2 *The Problem* I set the scene by investigating specific interaction challenges that users face when dealing with ubiquitous computing environments. I start with an historical account of ubicomp, the notion of context-awareness and common interaction patterns in ubicomp environments. I then provide an overview of the interaction challenges users face when dealing with these systems, and explore the notions of *intelligibility* and *control* as means to address these challenges. Finally, I position these challenges within Norman's Stages of Action model (Norman, 2013b), and clarify how they relate to the different chapters in this dissertation.

CHAPTER 3 *A Design Space for Intelligibility and Control* In this chapter, I explore design decisions faced by designers when adding support for intelligibility and control. I introduce a design space for intelligibility and control techniques that consists of six dimensions and allows for classification and comparison of different techniques. The following chapters introduce techniques that are situated at different points in the design space. Chapters 5–7 explore the *timing* dimension in the design space.

CHAPTER 4 *Exploratory Study of a Context-Aware Healthcare System* I describe an exploratory study of a proactive, context-aware mobile guidance system aimed at aiding nurses in their daily care-giving routines. The study provides additional

insights into the issues people face when interacting with context-aware systems, and allowed for a better understanding of the impact of different design choices.

CHAPTER 5 *Feedforward* I explore the design principle feedforward, which tells the user what the result of their action will be. Regarding the *timing* dimension, feedforward provides intelligibility *before* the action. I provide a new definition of feedforward and further disambiguate it from the related design principles *affordances* and *feedback*. In addition to the discussion of several existing examples of feedforward, I describe the *Feedforward Torch* prototype. Finally, I identify four new classes of feedforward: *hidden*, *false*, *sequential*, and *nested* feedforward.

CHAPTER 6 *Slow-Motion Feedback* Next, I present *slow-motion feedback*, a technique to provide intelligibility *during* the execution of system actions. Slow-motion feedback provides users with sufficient time to *notice* what the system is doing and *intervene* if necessary. I discuss *The Visible Computer*, an application of slow-motion feedback that uses projection to show in-place, real-time feedback about system actions that are being executed. I then introduce a design space to reason about the time at which feedback is provided. This design space is used to differentiate between slow-motion feedback, feedforward and feedback; and to analyse notable applications of slow-motion feedback.

CHAPTER 7 *Why Questions* In this chapter, I look into providing intelligibility and control *after* actions have been executed. I present *PervasiveCrystal*, a framework for building context-aware applications that support automatically generated answers to 'why?' and 'why not?' questions about their behaviour.

CHAPTER 8 *Intelligibility and Control for Proxemic Interactions* I describe a case study in supporting intelligibility and control for proxemic interactions. Proxemic interactions feature people-aware ensembles of devices that employ fine-grained knowledge of the identity, proximity, orientation or location of their users, and can thus be seen as a specific type of context-aware systems. I discuss the design and implementation of *Proxemic Flow*, a system that relies on a floor display to assist users in interacting with the primary proxemic-aware display. I describe several dynamic peripheral floor visualizations to inform users about tracking status, indicate action possibilities, and invite and guide users throughout their interaction with the primary display.

CHAPTER 9 *Conclusions* To conclude, I summarize the contributions of this dissertation and discuss future work.

2

THE PROBLEM: INTERACTION CHALLENGES IN UBIQUITOUS COMPUTING

In this chapter, we investigate the specific interaction problems that occur within ubiquitous computing environments, and what it is that specifically makes interaction within these environments difficult for end-users. We start with an analysis of Weiser's original vision of ubiquitous computing (Weiser, 1991), followed by examples of the types of interaction and user interfaces that are common in these environments, in particular focusing on context-aware computing. We then provide an overview of what changed with context-aware computing, and the different challenges users encounter when interacting with context-aware systems. We explore the notions of *intelligibility* and *control* as means to address these interaction challenges. Finally, we situate these interaction problems in Norman's Seven Stages of Action model, and clarify at what stages in the interaction with context-aware systems users face difficulties.

2.1 UBIQUITOUS COMPUTING – THE DAWN OF CONTEXT-AWARE COMPUTING

In his seminal Scientific American article (1991), Mark Weiser outlined his vision of *ubiquitous computing*, in which computing moves beyond the desktop and into our everyday environments. He described ubiquitous computing (ubicomp) as the third wave of computing, where each person has access to multiple computing devices that are integrated into their daily environments. In contrast, the first wave of computing is the mainframe era in which many people used a single computer, while the second wave of computing is the personal computer (PC) era, with one computer per person (Figure 2.1).

Weiser's vision has been influential for research in human–computer interaction, and in computer science in general. Given the proliferation of large interactive surfaces and mobile devices (e.g., smart phones, tablets and smart watches) together with the integration of computing into household appliances (e.g., smart TVs, thermostats[1] smart lighting), Weiser's prediction of multiple computers per person has certainly been realized (Bell and Dourish, 2007). Weiser also suggested the use of computing devices of different form factors. In the late eighties and early nineties (Weiser et al., 1999), Weiser developed a number of devices together with his colleagues at the Xerox Palo Alto Research Center (PARC) to illustrate this idea. They developed the inch-scale PARCtabs, the slightly larger foot-scale PARCpads (both mobile with wireless communication abilities), and yard-scale LiveBoards, shown in Figure 2.2. These three types of devices—'tabs', 'pads' and 'boards'—have now indeed become the dominant form of modern computing, currently instantiated

1 For example, the Nest thermostat: `https://nest.com/thermostat`

as smartphones, tablets and large interactive surfaces such as tabletops and public displays (Davidoff, 2012; Abowd, 2012).

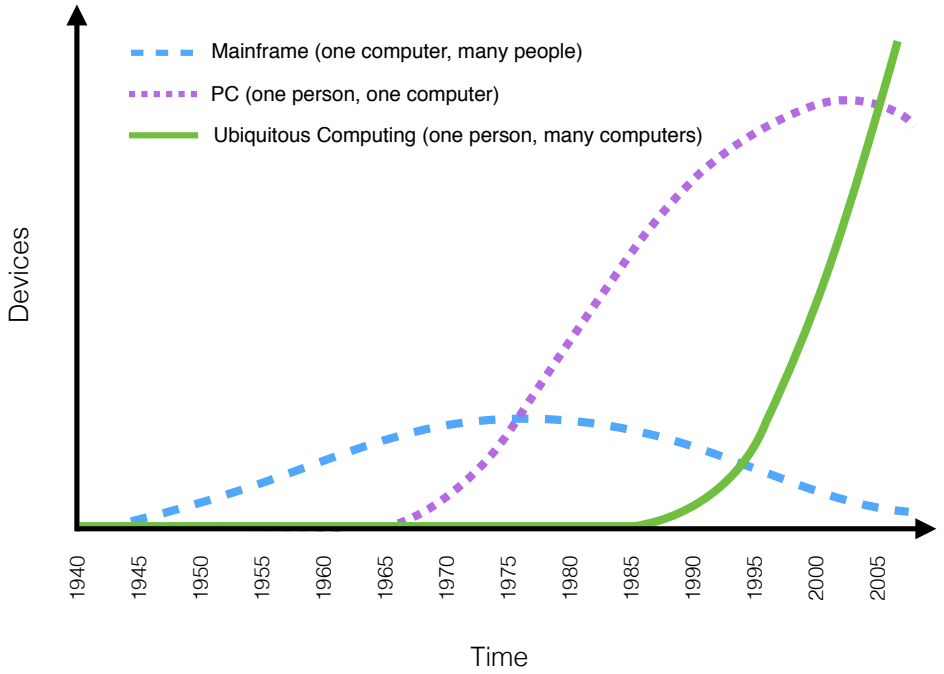


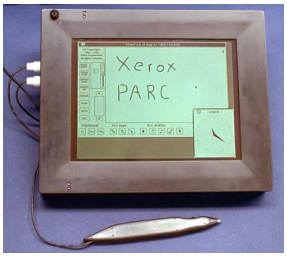
Figure 2.1: The three waves of computing according to Weiser: mainframes, PCs, and ubiquitous computing[2].

Weiser's article (1991) starts with the now famous phrase: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." He argued to push computers into the background, and making them an integral, invisible part of people's lives. In later articles, Weiser and Brown clarified their idea of 'the invisible computer' with the concept of *calm computing* (Weiser and Brown, 1995, 1997), where 'calm' refers to the desired state of mind of the user (Weiser et al., 1999). They argue that calm technology "will move easily from the periphery of our attention, to the center, and back" (Weiser and Brown, 1995), and offers information to users, without demanding attention. In other words, the computer is invisible unless attention from the user is needed.

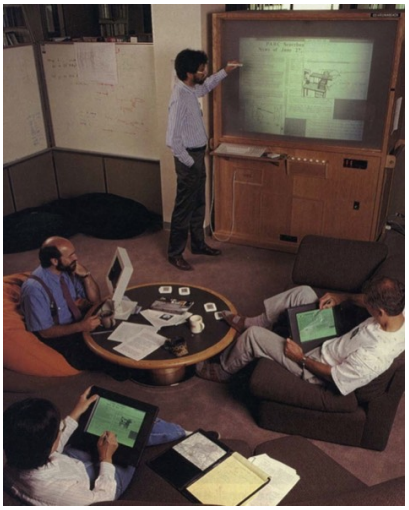
With his vision of ubiquitous computing, Weiser laid the foundations for *context-aware computing* (Schilit et al., 1994). Context-aware systems use sensors to gain knowledge about the context—the situation in which they are used—and try to adapt to this context. Having systems react according to changes in the context of use without requiring explicit input is a way to effectively move computers into the

2 Reproduced from (Want, 2009) and Mark Weiser's personal website: `http://www.ubiq.com/hypertext/weiser/UbiHome.html`

(a)



(b)



(c)

Figure 2.2: Weiser's vision of three different device form factors: (a) the inch-scale PARCtab, (b) the slightly larger foot-scale PARCpad, and (c) the yard-scale LiveBoard (image sources: Weiser (1991) and PARC[3]).

background. Context-awareness can therefore be seen as an enabling technology for realizing Weiser's vision of ubiquitous computing (Abowd and Mynatt, 2000).

Schilit et al. (1994) state:

> "Such context-aware systems adapt according to the location of use, the collection of nearby people, hosts and accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment. Context includes lighting, noise level, network connectivity, communication costs, bandwidth, and even the social situation; e.g., whether you are with your manager or with a co-worker."

Later, Dey (2001) further refined context as "any information that can be used to characterize the situation of an entity" where an entity is "a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". Dey considers a system context-aware if "it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task" (Dey, 2001).

The idea is that context-aware systems would release us from the burden of explicitly providing input to perform the desired operations and from having to pay full attention to the system. An example of this vision is the work by Cooperstock et al. (1997) on *reactive environments*, in which "the technology itself, rather than a human, manages the low-level operation of the room". This is similar to Buxton's

3 http://blogs.parc.com/blog/2010/09/its-time-to-reap-the-context-aware-harvest/

notion of 'off-loading' secondary commands to the system (Buxton, 1995b). Cooperstock et al. argue for reactive environments that respond to the user's high-level actions, thereby reducing their cognitive load and the amount of training required. For example, in the reactive videoconferencing room, slightly leaning left causes a motorized camera to pan. In contrast with traditional computing systems, context-aware systems take action based on input collected from the environment using sensors, allowing for the use of *implicit interaction*. As argued by Schmidt (2000), this represents a radical shift in human–computer interaction from direct manipulation graphical user interfaces towards more implicit interaction that is based on the situational context. Earlier, Nielsen (1993) already identified this shift towards more implicit interaction with next-generation 'non-command' interfaces that "allow users to focus on the task rather than on operating the computer". One of the arguments for more implicit interaction, as argued by Nielsen (1993), is that "many users would probably prefer a computer that did what they actually wanted, rather than what they *said* they wanted". As an example, Nielsen discusses the Portholes system (Dourish and Bly, 1992) that aimed to provide lightweight awareness in distributed work groups using a collage of video snapshots of co-workers that are updated every 5 minutes (Figure 2.3). Nielsen mentions that with Portholes, users "do not need to take any action to inform their co-workers that they are in the office or that they are meeting with somebody and should not be disturbed".

In her dissertation, Ju (2008) discusses a number of key differences between implicit and explicit actions, including *attention*, *exclusivity*, and *grounding*. With respect to attention, she mentions that explicit actions require the user's full attention, while implicit actions occur in the user's attentional periphery. In contrast to explicit actions, which exclude other focal targets, implicit actions are non-exclusive. Finally, regarding grounding, Ju states that implicit actions require some level of interpretation, while explicit actions tend to have a generally understood meaning. Implicit input occurs when a user is, perhaps unconsciously, performing actions in their attentional periphery and the system reacts accordingly (e.g., walking past a public display), whereas explicit input is a focused action by the user to provide input to the system such as clicking a button (Ju, 2008).

Early research in context-aware computing mainly focused on *location-awareness*, i.e., having systems respond to location changes. Schilit et al. (1994) discuss several techniques that make use of the ActiveBadge indoor location tracking system (Want et al., 1992), such as selecting devices based on physical proximity (e.g., the printer in the current office), easily exchanging information with nearby devices and people and location-aware reminders. Similar functionality is commercially available today, e.g., Apple's AirPlay feature to share information across devices[4] or Google Now, which provides location-specific information and reminders on mobile devices[5]. Other examples of location-aware systems are context-aware mobile guides such as GUIDE (Cheverst et al., 2000), CyberGuide (Abowd et al., 1997) or ImogI (Luyten et al., 2004); satellite navigation systems such as TomTom[6]; and mobile map

4 https://www.apple.com/airplay/
5 http://www.google.com/landing/now/
6 http://www.tomtom.com/

Figure 2.3: An example of the shift towards more implicit interaction: the Portholes system (Dourish and Bly, 1992), providing awareness of distributed groups of co-workers through video snapshots (image source: Buxton, 1995a).

apps such as Google Maps[7]. The implicit interaction aspect of these location-aware systems lies in the fact that the user just moves (or drives) to another location, to which the system adapts its contents. A satellite navigation system, for example, would provide updated turn-by-turn directions, while a mobile guide would show updated location-specific information. Users do not need to explicitly provide their location to the system, or request information that is relevant to their current location; their only 'command' to the system is their physical presence at that location (Nielsen, 1993).

Researchers also explored the use of other sensors, such as accelerometers to sense device orientation and switch between portrait and landscape mode (Hinckley et al., 2000), the use of RFID tags and scanners to link physical objects to their virtual counterparts (Want et al., 1999) or RFID-enabled badges to identify people in front of a public display and adapt the contents accordingly (McCarthy et al., 2004). In addition, researchers have built 'context-aware environments' or 'smart spaces' using frameworks that linked several devices and context-aware services together: e.g., HP Cooltown (Kindberg et al., 2002), the Stanford iRoom (Borchers

7 https://www.google.com/maps

et al., 2002), and Speakeasy (Newman et al., 2002). Examples of such context-aware environments are meeting rooms (Cooperstock et al., 1997), classrooms (Abowd, 1999) and smart homes (Kidd et al., 1999; Mozer, 1998; Cook et al., 2003).

However, the characteristics of context-aware computing (such as the use of implicit interaction) also introduce new interaction challenges, which are only exacerbated when we move from simple context-aware applications to context-aware environments. In what follows, we provide an overview of what exactly changed with context-aware computing, and discuss how these changes result in interaction challenges.

2.2 INTERACTION CHALLENGES WITHIN CONTEXT-AWARE COMPUTING

2.2.1 *What Changed with Context-Aware Computing*

There are several basic notions about the way we interact with computers—and with GUI/WIMP systems (Hutchins et al., 1985) in particular—that changed with the introduction of context-aware computing (Bellotti et al., 2002). Compared to interacting with context-aware environments, interaction with desktop computers and GUI applications is easy (Bellotti et al., 2002). There is no doubt what the system is: (the box on your desk), there is no confusion over how to provide input to the system (either using the mouse or keyboard) and where feedback will appear (on the display). Unlike the adaptive and dynamic context-aware system, the GUI/WIMP system is fairly predictable; it typically does not do anything unless it is instructed to do so by the user.

Nielsen (1993) describes several dimensions in which next generation 'non-command interfaces' (which includes context-aware computing) differ from traditional interfaces. Similar to the idea of reactive environments (Cooperstock et al., 1997), he mentions that in these interfaces most of the control will pass from the user to the computer, which will "sometimes even choose to perform actions without explicit user control". The computer's role will shift towards "interpreting user actions and doing what it deems appropriate". In addition, the 'locus of interaction' will not be on the computer's screen, mouse and keyboard, but will be "embedded in the user's environment, including the entire room and building" (Nielsen, 1993).

Basic interaction mechanisms that we learned over time tell us where input will go (e.g., a flashing cursor), and what we can do (e.g., menus that provide visibility of the system's functionality and encourage exploration). Bellotti et al. (2002) argue that there are no such interaction mechanisms designers can rely on when designing 'sensing systems' (i.e., systems that gather input through sensors). Given the shift in the 'locus of interaction' to the user's environment (Nielsen, 1993), how do users know where the system is, and how they can address it (Bellotti et al., 2002)? Consider, for example, the case of a smart home: how do users receive feedback about actions taken by the home (e.g., turning off the heating), and what caused the system to take these actions? How can inhabitants override the home's automatic behaviour, or specify deviations from inferred routines (Mennicken et al., 2014)?

In recent years, there has been a shift from the GUI/WIMP paradigm to more natural touch- and gesture-based user interfaces as employed by smartphones and tablets. Still, the touch-based interaction style shares many commonalities with the GUI/WIMP paradigm. It is essentially an evolved, easier to use implementation of direct manipulation, in which users directly touch objects of interest using their fingers instead of using a mouse and pointer. The issues highlighted in the papers by Bellotti et al. are thus still valid today. Moreover, one could argue whether natural user interfaces (NUIs) are really natural (Norman, 2010). Gestural and speech interfaces, for instance, exhibit many problems related to discoverability, visibility and feedback.

In summary, we argue that there are four main changes when we move from traditional systems to context-aware systems. Each of these changes result in interaction challenges for the user that are not (or only marginally) present in traditional systems:

- *Dynamic behaviour*: The same input can give different results depending on the context, making systems less predictable (Ju, 2008). How do users know that the system is doing, has done or will do the right thing (Bellotti et al., 2002)?
- *Implicit Input*: Users might be unaware that the system recognizes their actions and interprets those actions as input to the system (e.g., when walking past a public display that reacts to the presence of people). In the worst-case scenario this can result in the Midas touch problem, where every (unconscious) action by the user is interpreted as another command (Jacob, 1990). How can users know when they are addressing the system, and avoid doing so unintentionally (Bellotti et al., 2002)?
- *Autonomy*: Systems may react to context changes by taking autonomous actions (Cooperstock et al., 1997). How can users be made aware of the system state, and know what caused these autonomous system actions to occur? How can users intervene when the system takes an undesired action? Can an action in progress be cancelled or controlled?
- *Complexity*: Context-aware systems may take decisions based on complex rule-based systems or rely on machine learning algorithms, which can be hard to understand for end-users (Dey and Newberger, 2009; Tullio et al., 2007). How can users build up a mental model of the system, and be made aware of the general reasoning employed by the system, without being overwhelmed by technical details or possibly contradictory information (Norman, 2013b, pg. 183)?

Next, we delve deeper into these interaction challenges and also discuss a number of criticisms of context-aware computing.

2.2.2 *Fundamental Problems with the Notion of Context-Aware Computing*

Context-aware computing has long been hailed as the solution to interaction challenges and usability problems with computers. The general idea was that if we add more sensors, more data and better inference algorithms, our systems would automatically be able to do 'the right thing' (Taylor, 2006). Taylor (2006) argues that this vision is exemplified by early work on context-aware computing, such as the omission of "any representation of human involvement in sensing, interpreting and acting on contextual information" in the conceptual framework and toolkit introduced by Dey et al. (2001).

Given that computing systems are not (yet) able to read our minds, the outset of fully autonomous systems is likely to be infeasible. Indeed, as argued by Bellotti and Edwards (2001), there are several aspects of the context that cannot be reliably sensed by computers. Rogers (2006) even compares the challenge of making computers do the right thing automatically to the promise of *strong AI*, "a vision in which a computer is not merely a tool, but a mind in itself". If we would succeed in doing this, we would therefore also have solved all the difficult intricate challenges in the field of AI, and would have essentially created a human-like intelligence, which will probably remain an unsolved problem for many years to come.

A similar argument is made by Erickson (2002), who argues, "the context-awareness exhibited by people is of a radically different order than that of computational systems. People notice a vast range of cues, both obvious and subtle, and interpret them in light of their previous experience to define their contexts." In contrast, context-aware systems only detect a very small set of cues (based on the available sensors), and have limited means to reason about these sensor values. Doing the right thing would require considerable intelligence. The example given by Erickson is that of a phone's awareness of being motionless in a dark place with high ambient noise, which is still very different from the human awareness of being in a theatre (Erickson, 2002). Even though one might argue that services such as Foursquare[8] and Google Maps, which have an extensive set of venues at specific locations, could infer that the user is in a movie theatre based on their current location, this kind of context-awareness would still fail when the user is watching a movie at an outdoor festival, or at home with friends, or when the movie theatre is repurposed for other events. It is exactly this interpretation of subtle cues that proves difficult for machines, no matter how much data or sensor readings we throw at them. For example, a recent study of the Nest smart thermostat (Yang and Newman, 2013) found that although the thermostat took into account users' inputs, it would often fail to catch their real intent. Participants often felt that it made incorrect assumptions and felt out of control as they could not train it to make other assumptions. One participant even described the Nest as being 'arrogant', "feeling that it would do whatever it thought was right" (Yang and Newman, 2013). These observations clearly demonstrate Erickson's point on the difficulty of sensing the more subtle cues that are part of human situational awareness.

8 https://foursquare.com/

Greenberg (2001) states that context is "a dynamic construct viewed over a period of time, episodes of use, social interaction, internal goals and local influences". Or in other words, context "is a continually evolving and highly situation-dependent construct". Greenberg claims that the idea of using context as a stable set of contextual states that can accurately be inferred from a predefined set of sensed information is—in all but the simplest of cases—difficult or even impossible. He supports this claim by reviewing several existing theories that suggest the dynamic nature of context, such as situated actions (Suchman, 1987) and activity theory (Nardi, 1995). What follows from this is also that a correct 'set of rules' that determines the appropriate action given a specific context cannot always be defined. Finally, Greenberg (2001) says that it is all too easy to trivialize context when representing it in a system, and that as a result, there is a strong likelihood that the system might get things wrong and take inappropriate actions. Moreover, he suggests that systems should be conservative, and only take risky actions when there is "compelling evidence of correctness", since a single inappropriate action may be enough to preclude people from using the system further.

Dourish (2004) discusses how, even though context-aware computing has been inspired by social scientists' critiques on conventional system design (i.e., Suchman's work on how interaction with systems cannot be separated from the setting in which it occurs (Suchman, 1987)), "the social and technical ideas often sit uneasily together". Dourish mentions that most of the ubicomp literature has tackled context as a *representational* problem (i.e., "what is context and how can it be encoded?"). According to Taylor (2006), Dourish instead argues that, "context is something that is continuously being made and dependent to a large degree on the ever-changing relations between people and the resources they bring to bear in everyday settings". Context is thus an *interactional* problem (Taylor, 2006). This is similar to Greenberg's critique about how context is not a stable set of contextual states, but rather a dynamically evolving situation-dependent construct (Greenberg, 2001). Dourish observes that the sociological critique on context-awareness is that "the kind of thing that can be modelled" using the representational approach to context, "is not the kind of thing that context is" (Dourish, 2004).

Next to these high-level critiques on the notion of context-aware computing, a number of researchers have also reported on their experiences and issues encountered with real-world deployments of context-aware systems. Based on their experiences with the GUIDE location-aware mobile guide, Cheverst et al. (2001) discuss several pitfalls that designers should take into account when developing context-aware systems. First, they discuss the difficult balance in providing 'ease of use' by adapting to the current context and still maintaining sufficient freedom or flexibility for the user. For example, an initial version of GUIDE only allowed users to receive touristic information about the area they were currently in. Since this lack of flexibility caused frustration, Cheverst et al. later added the option to search information based on keywords as well. As another example, an earlier version of GUIDE removed all closed attractions from the list of nearby attractions, which again frustrated some users. Secondly, visitors using GUIDE also struggled when the context information was not accurately sensed; when the system experienced difficulties

in correctly determining the user's location, some locations that were presented as nearby attractions were, in reality, not so nearby as users expected. Finally, Cheverst et al. (2001) suggest using sensing technology that is dependable, meaning it is both accurate and available in a timely manner (e.g., early GPS antenna's only obtained a GPS fix after a couple of minutes). Moreover, they state that there needs to be a way for users to override the system's context adaptation strategy.

One could argue that context-awareness in our daily environments has mostly been confined to specific, predictable and low-risk microinteractions (Saffer, 2013), such as a smartphone's proximity sensor that detects when the user is holding the device close to their face after which the device blanks the screen and ignores touch screen input, the use of accelerometers to automatically rotate the screen (Hinckley et al., 2000), or lights equipped with motion sensors that automatically turn on or off depending on people's presence (Cooperstock et al., 1997). Even in those simple cases, the system might still perform the wrong action based on the sensed information and consequently frustrate its users—e.g., when a smartphone's display stays blank because part of the user's hand is triggering the proximity sensor, or when a mobile device inadvertently switches to landscape or portrait mode when laying on a flat surface.

Context-aware systems that act based on simple context rules—in other words, a 'trivialized notion of context' (Greenberg, 2001)—may feel like magic to users when they perform the right action at the right time. However, when they fail, the consequences can be severe and even lead to users abandoning the system altogether (Greenberg, 2001; Vihavainen et al., 2009). In those situations, rather than feeling like magic, the technology may seem haunted instead.

2.2.3 *The Need for Intelligibility and Control*

In this section, we explore solutions to address the previously discussed problems with context-aware computing. In particular, we focus on two principles that were proposed as key features that should be supported by context-aware systems: *intelligibility* and *control*.

2.2.3.1 *Intelligibility: Helping Users Understand How the System Works*

In an ideal world, in which the interpretation of sensed context information would be 100% accurate, computers would essentially be rendered invisible. Users would not 'notice' the computers embedded in their environment, they would just experience the right actions 'magically' being performed at the right time. As discussed before, this assumption is, however, unrealistic. Bellotti and Edwards (2001) argue that "the more we try to get systems to act on our behalf, especially in relation to other people, the more we have to watch every move they make." Given that context-aware systems cannot perfectly sense the user's context, Bellotti and Edwards propose that context-aware systems should be made intelligible[9]: they should be able

9 The notion of intelligibility was first introduced by Brown and Newman (1985).

to "inform users about what they know, how they know it, and what they are doing with that information".

Analogously, Dourish (1995) discussed how systems could provide 'accounts' of their own behaviour through computational reflection—essentially a self-representation—that helps the user in understanding how the system works. An account not only describes behaviour, but also provides users with the means to control that behaviour. Dourish stresses that accounts are not the same as mental models (Norman, 2013b), as they exist on different sides of the interface: "We distinguish between an account of system behaviour as offered by a system, and the understanding of system behaviour formed by a user in response" (Dourish, 1995). Dourish argues that even though abstractions in software are useful, they sometimes hide details that turn out to be crucial to our interactions. In a similar fashion, hiding the context inference process or the rules of behaviour can pose problems to users when context-aware systems fail. Taylor (2006) stresses that work that is 'off-loaded' to the system (Buxton, 1995b) and is directly related to the users' interaction with information "should be easily understood by users".

In his critique of the representational view on context-aware computing (Dourish, 2004), Dourish proposes to think instead about "how ubiquitous computing can support the process by which context is continually manifest, defined, negotiated and shared". As an example, he explains how "a system can display aspects of its own context—its activity and the resources around which that activity is organised". By doing that, the system essentially provides users with "a more nuanced interpretation of the meaning of the system's action". Correspondingly, Taylor (2006) discusses the need for design principles that enable users to make their own inferences about the state of the world "as perceived by them, and as reported by the system".

Similar arguments have also been made by Cheverst et al. (2005) and Kay et al. (2003); Assad et al. (2007). They suggest that in order not to surprise or frustrate the user, context-aware systems should allow the user to interrogate or scrutinise their user model (which includes the rules of behaviour of the system). Greenberg (2001) mentions that "People should be able to see what context the system thinks it has inferred." Abowd and Mynatt (2000) further discuss the issues with emphasizing 'invisibility' in Weiser's vision of ubiquitous computing, and how it conflicts with informing users about how they are being sensed. They also stress the need for making this information visible, and as a next step, providing users with control over what the system is doing. Similarly, Cooperstock et al. (1997) mention that appropriate feedback and opportunities for user intervention are necessary prerequisites to realize their vision of reactive environments. Rehman et al. (2002) discuss a number of issues introduced by interfacing with 'the invisible computer', such as the user's lack of an appropriate mental model of the system which results in difficulties with predicting the system's behaviour or even its available features. They also state that users often have no means to control or override the system, due to either feasibility concerns ("what control interface should be offered to users?") or the desire to 'hide' the computer.

Coutaz et al. (2005) state that one of the major challenges for context-aware systems is "finding the appropriate balance between implicit and explicit interaction", and "the appropriate degree of autonomy, which can impact every level of abstraction". In a later article, Coutaz (2007) presents the notion of 'meta-user interfaces', a set of functions and user interfaces necessary to allow users to evaluate and control the state of interactive ambient spaces.

Edwards and Grinter (2001) discuss seven challenges for introducing ubiquitous computing into the home (or in other words, challenges for 'smart homes'). One of their challenges is "The 'Accidentally' Smart Home", in which they discuss issues of predictability and complexity, and what to do when the context-aware adaptation fails. The problem is essentially "intelligibility in the face of radical—and perhaps unexpected—connectivity" (Edwards and Grinter, 2001). Edwards and Grinter argue that context-aware systems, especially those that exhibit an intermediate level of intelligence by inferring the user's context and taking action accordingly, will never be right all the time and will thus need to provide models of how they arrive at their conclusions to users. Inference should be "done in a way that is predictable, intelligible, and recoverable" (Edwards and Grinter, 2001).

Edwards and Grinter (2001) further discuss important design questions that need to be addressed in order to solve that challenge, such as what kinds of affordances should be provided to make the system intelligible, how to inform inhabitants about the potential configurations of different devices in the home and when these devices are interacting, where the locus of interaction will be, or how they can control these devices and the home in its entirety. These questions were further explored in a later paper by Bellotti et al. (2002), in which it is argued that such sensing systems are hard to use because designers of these systems have to reinvent all the user interface mechanisms that we rely on everyday in graphical user interfaces, such as appropriate feedback (Norman, 2013b) and feedforward (Vermeulen et al., 2013b; Djajadiningrat et al., 2002; Wensveen, 2005) (see Chapter 5).

2.2.3.2 *Control: Allowing Users to Recover from Mistakes*

Next to intelligibility, users should also be provided with the means to override and *control* context-aware systems when they take inappropriate actions or make a mistake. As argued by Greenberg (2001), "some contexts can never be inferred" (e.g., because they appear only infrequently), so users should have the option to completely override the system. Failing to provide users with control might result in users who lose trust in the system (Barkhuus and Dey, 2003) or even abandon it altogether (Greenberg, 2001; Vihavainen et al., 2009).

In a study of a location-aware microblogging service, Vihavainen et al. (2009) found that users were worried about automatic location sharing, and expressed concerns such as not being able to lie anymore about their whereabouts. In one group of users, inappropriate automation led to abandonment of the service. Moreover, the study also revealed problems related to intelligibility: users were at times confused about (perhaps incorrectly) reported locations, as they had no means to assess the system's confidence in that location.

Experiments with the MavHome (Cook et al., 2003), a smart home that adapted to the routines of inhabitants over time, exemplify the problems resulting from a lack of user control. When an occupant moved into the home, who, unlike previous occupants, led a more chaotic life and often had visitors over late at night, the home failed to adapt from the previously learned behaviour. The lights would still be turned off every evening, despite the fact that the occupant was awake and active. A striking finding of the study was that the occupant later stated, "they were learning to live in the dark because it was too bothersome to correct the system" (Youngblood et al., 2005). Moreover, the study also reported on a number of fights between the same inhabitant and the home over control.

A number of researchers have observed that users' sense of control tends to decrease when the autonomy of the application increases (Barkhuus and Dey, 2003; Cheverst et al., 2002; Vihavainen et al., 2009). Sheridan and Parasuraman (2005) introduce a scale of 8 different levels of automation ranging from "the computer offers no assistance, the human must do it all" to "the computer selects the method, executes the task, and ignores the human", and argue that the appropriate level of automation depends on the situation. Similarly, Bellotti and Edwards (2001) discuss several design strategies for control depending on the system's confidence in what the desired outcome is, such as being offered a means to correct the system's action, a means to confirm the system's action, or offering the user available choices for system action. In addition, Greenberg suggests to take into account the amount of risk involved, i.e., whether taking action might have 'dangerous consequences': "context-aware systems should be fairly conservative in the actions they take, should make these actions highly visible, and should leave 'risky' actions to user control" Greenberg (2001).

While not specifically targeting context-aware systems, Don Norman wrote on how appropriate design can help us deal with automation problems (Norman, 1990). Norman mainly uses examples from automation in the aviation industry (e.g., the autopilot), but we believe that his arguments are also applicable to context-aware systems in general, and in particular, to automation in context-aware environments such as smart homes (Mennicken et al., 2014). Norman's view is that it is not over-automation that causes problems, but the lack of appropriate feedback. He states that the source of all difficulties is that "automation is powerful enough to take over control that used to be done by people, but not powerful enough to handle all abnormalities". Essentially, this is similar to the arguments made by Bellotti and Edwards (2001) or Erickson (2002): there are some situations in which the system either cannot make a decision on what to do next, or will have an increased chance of making the wrong decision and thus needs to defer to the user instead. In order to allow users to take over, we need to offer them the means to do so (i.e., *control*) and provide them with sufficient information about what the system has done and is currently doing (i.e., *intelligibility*) to allow for intervention. Similar to the context-awareness in reactive environments (Cooperstock et al., 1997), automation in airplanes has the goal of reducing the workload to allow the operator to focus on what's important (and, in the case of the aviation industry, increase safety). However, Norman argues that automation also (mentally) isolates the crew from

the details of flying and the state of the aircraft, and therefore, when the equipment fails, increases the difficulties and the magnitude of the problem in diagnosing the situation and determining the appropriate course of action. It is this lack of engagement, according to Norman, that causes the main problems with automation: "feedback is essential because equipment does fail and because unexpected events do arise". He suggests that what is needed, is "continual feedback about the state of the system, in a normal natural way", i.e., without overwhelming or interrupting the user. Norman's viewpoint is related to the arguments made by Dourish (1995) about how abstractions hide details that are sometimes crucial to our interactions.

Intelligibility or 'scrutability' can be seen as a prerequisite for control (Kay et al., 2003; Assad et al., 2007; Cheverst et al., 2005) or as the most basic level of control (Coutaz, 2007; Bellotti and Edwards, 2001). To clarify this, consider the case of a very simple context-aware system: an outdoor light equipped with a motion sensor. Our basic understanding of the light's behaviour, i.e., that the light turns on when it detects motion and that there is a certain amount of time after which it automatically turns off, together with the location and range of the motion sensor allows us to modify our behaviour to control it. Because we know that the light reacts to motion, we would typically move into the range of the motion sensor and wave our hands until the light turns on again. Similarly, intelligible systems that provide us with an understanding of how they work, allow us to infer what actions are needed to influence their behaviour. Coutaz (2007) defines several levels of control, ranging from observability (which she considers the minimum level that should be supported), over traceability to controllability. Observability refers to allowing users to "evaluate the internal state of the service from its current perceivable representation", while traceability enables them to "observe the evolution of the service over time". Both levels of control can be considered to be aspects of intelligible systems.

From this overview, it is clear that we need support for *intelligibility* and *control* to unlock the potential of context-aware environments and make such smart spaces "usable, predictable and safe for users" (Bellotti and Edwards, 2001). However, this is not an easy problem to solve. Studies have revealed that users have little interest in learning about a system's workings as an independent activity (Yang and Newman, 2013), and their preconceived notions about how a system works may be difficult to change (Tullio et al., 2007). Moreover, a study by Lim and Dey (2011b) found that intelligibility might have a negative impact on the user's impression of the system. They conclude that intelligibility is helpful for applications with high certainty, but harmful if applications behave appropriately but display low certainty. There is also a delicate balance in the amount of detail that should be provided to users, where too much information might easily confuse them (Kulesza et al., 2013). Indeed, the use of intelligibility in commercial recommender systems, has mostly been confined to very high-level information, which could indicate that users are generally not interested in the low-level details. For example, Gmail's priority inbox uses high-level phrases to inform users of why it marked an email as important, such as "because of the sender of the message", "because of the words in the message", or "because you often read messages with this label". Similar high-level phrases are used in explaining recommendations by YouTube, Amazon or the Apple App Store (e.g., "because

you bought/liked/watched X"). It is, however, unclear whether these high-level explanations work well because recommender systems mostly 'suggest' rather than 'decide', and if this would also generalize to other systems with a higher degree of autonomy (see also Bunt et al., 2012). Especially with autonomous systems, care should be taken not to overly dumb down the information that the system provides, to avoid the abstraction and mental isolation problems discussed by Dourish (1995) and Norman (1990).

2.3 INTELLIGIBILITY AND CONTROL IN NORMAN'S STAGES OF ACTION

In this dissertation, we explore the problem of supporting intelligibility and control in context-aware systems from a design perspective. This dissertation serves as a design space exploration, in which we aim to be both generative by guiding designers in exploring various ways to support intelligibility and control, and generalizable by exploring techniques that can be applied by interaction designers in a wide range of ubicomp scenarios. Given this focus, we now revisit the previously discussed interaction challenges (Section 2.2) and situate these within a design model that has been widely used for analysing interaction problems: Don Norman's Seven Stages of Action (Norman, 2013b). We start by briefly discussing the basic flow of Norman's model including the gulfs of execution and evaluation.

2.3.1 *Norman's Seven Stages of Action*

Norman introduced the *Action Cycle* in his book "The Psychology of Everyday Things" (often abbreviated as 'POET') (Norman, 1988) as a way to analyse how we interact with 'everyday things', including doors, light switches, kitchen stoves but also computers and information appliances (e.g., mobile devices). The book was later updated with a new title and clarified terminology (Norman, 2002) ("The Design of Everyday Things" or 'DOET') and recently received a major revision (Norman, 2013b).

Norman suggests that there are two parts to an action: executing the action and evaluating the results, or "doing and interpreting" (Norman, 2013b). Furthermore, actions are related to our goals; we might formulate a goal, execute certain actions to achieve that goal, evaluate the state of 'the world' to see whether our goal has been met, and if not, execute more actions to achieve our goal, or otherwise, formulate new goals that again result in more actions (Figure 2.4). He introduces the *Stages of Execution* and the *Stages of Evaluation* as a break-down of these two parts, which together with goal formulation, form the *Seven Stages of Action*. Starting from our goal, we go through three stages of execution: plan (the action), specify (an action sequence) and perform (the action sequence). To evaluate the state of the world, there are again three steps: perceive (what happened), interpret (making sense of it) and compare (was what happened what was wanted?), as illustrated in Figure 2.4. Norman notes that not all activity in these stages is conscious (even goals may be subconscious), "We can do many actions, repeatedly cycling through the stages

while being blissfully unaware that we are doing so. It is only when we come across something new or reach some impasse, some problem that disrupts the normal flow of activity, that conscious attention is required." (Norman, 2013b, pg. 42).

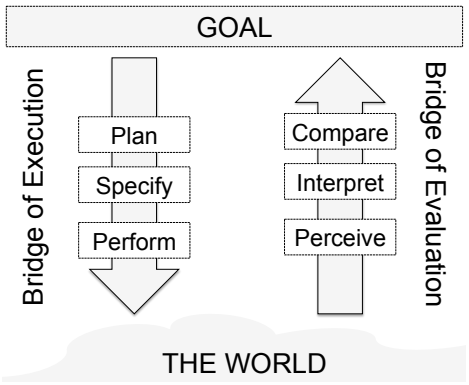


Figure 2.4: Norman's Action Cycle (Norman, 2013b): formulating goals, executing actions that impact the state of 'the world', and evaluating these changes to see whether the goals have been met. The Seven Stages of Action consist of one stage for goals, three stages for execution and three for evaluation.

The main notion put forward in Norman's book is that "both execution and evaluation require understanding: how the item works and what results it produces", an understanding that can be communicated through design (Norman, 2013b). People face two essential problems when trying to use something: figuring out how it operates, and figuring out what happened. When we face difficulties in figuring out how something operates, we need to bridge the 'gap' between our goals and the realization of these goals which Norman calls *the Gulf of Execution* (the left side of Figure 2.4). On the other hand, when we have problems in trying to figure out what happened, there is a gap between the results of our actions in the world, and deciding whether our goals have been achieved, which Norman calls *the Gulf of Evaluation* (the right side of Figure 2.4). When we can successfully move between the stages of execution to perform our actions, and when we can successfully go through the stages of evaluation to assess whether our goals have been met, both gulfs have been bridged. It is the role of the designer to help people bridge the two gulfs (Norman, 2013b, pg. 38).

We now explain how Norman's Stages of Action can be related to context-aware systems. As discussed before, there are four major changes when we compare context-aware systems to traditional systems: dynamic behaviour, implicit interaction, autonomy and complexity. We investigate each of these within the context of Norman's framework.

2.3.2 *Interaction with Autonomous Systems*

Context-aware systems have the ability to act autonomously (e.g., based on information gathered from sensors), without user involvement. Traditional systems bridge the gulf of evaluation by employing informative feedback. Feedback typically confirms that the system has recognized and is responding to the user's actions, or shows what the effect of the user's actions was. When a system is acting on its own, there is no user action to which it responds. In this section, we will explore how to represent this type of system using Norman's framework, and how the gulf of evaluation can be bridged in this case. It is important to note that with autonomous systems, mistakes made by the system need to be accounted for in addition to mistakes made by users.

Norman (2013b, pg. 42–43) mentions that, "the action cycle can also start from the bottom, triggered by some event in the world, in which case we call it either data-driven or event-driven behaviour." This is precisely what happens when a context-aware system takes action on the user's behalf—or at least when the effect of that action is visible to the user. In this situation, we start on the right side of the action cycle (Figure 2.4), where something happens in 'the world', after which the user goes through the three stages of evaluation (perceiving the change, interpreting what they perceived and comparing that to their goals), as shown in Figure 2.5.

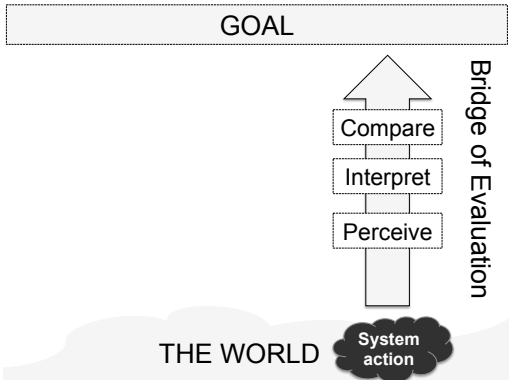


Figure 2.5: When a context-aware system acts autonomously, we start at the right side of the action cycle, where the user goes through the three stages of evaluation.

Unfortunately, autonomous systems do not always allow users to perceive that something has happened, or may fail to inform users of their current state, i.e., what the consequences are (Norman, 1990). If the result of the system's action is invisible, users have no way to detect potential mistakes. Even if systems do provide feedback about actions going on in the background, we cannot expect people will always pay close attention. As an example, Norman discusses an accident with a cruise ship that was caused by a failure in the ship's GPS antenna (2013b, pg. 215). The navigation system— which normally relies on an accurate GPS location—had, given the

antenna problem, switched to 'dead reckoning' mode[10], which the captain failed to notice. The problem here was that this mode switch was only indicated by the tiny letters 'DR' on the captain's display. In his book "The Design of Future Things" (2009, pg. 138), Norman notes, "If the inappropriate use of feedback is frustrating with simple devices such as elevators, what will it be like with the completely automatic, autonomous devices of our future?" Without proper feedback, in addition failing to notice the action, users may also be frustrated when the system suddenly takes an undesired action. Consider even the simple example of motion sensors in a smart home that result in all the lights getting turned on when you get in late at night, even though someone is sleeping on the couch in the living room.

Ju (2008) suggests that an autonomous system must "demonstrate how it will act, and also hint at what the consequences of its actions will be". In the case of the cruise ship, a demonstration of the mode switch together with information about its consequences might have succeeded in alerting the captain. Furthermore, providing feedback only after the action has already been completed prevents users from reacting when the action was undesired. In a similar argument, Norman (2009) discusses the tension for control between the system and the user. He mentions that there are bound to be mistakes due to the limited intelligence of the system (see also Section 2.2.2), and argues that many systems can make the problem even worse by making it very hard to recover from errors: "When the machine intervenes, we have no alternatives except to let it take over: 'It's this or nothing,' they are saying, where 'nothing' is not an option." (Norman, 2009, pg. 3)

This brings us to the next challenge users face when dealing with autonomous systems: having the means to control the system when necessary. As mentioned before, users go through the three stages of evaluation when the system takes action on their behalf to notice what happened and determine whether or not this matches their goal. Intelligible systems let users know what they are doing, but what happens when the system's action does not match the user's goal (e.g., the system that turns on the lights in the living room in the previous example)? In this case, the system should provide users with the option to recover from the mistake, or even prevent the mistake from ever happening by first demonstrating what it *will* do. The smart home could allow users to prevent the lights from turning on, by first slowly increasing the intensity of the lights, and providing a means to cancel this action (e.g., clicking one of the light switches). In both cases, there is again an execution phase involved to control the system. After perceiving the system's action and deciding that it does not match our goal, we formulate a new goal: overriding the system action. To reach that goal, we again go through the three stages of execution, as shown in Figure 2.6. In other words, systems should not only show users what they are doing, but also make clear how the system action can be overridden, and offer users control.

In Chapter 6, we introduce *slow-motion feedback* (Vermeulen et al., 2014), a technique that allows users to notice actions taken by the system, and provides them with sufficient time to intervene when the action is undesired. Making users aware

10 Dead reckoning means that the location is approximated based on the last known location, the current speed and direction.

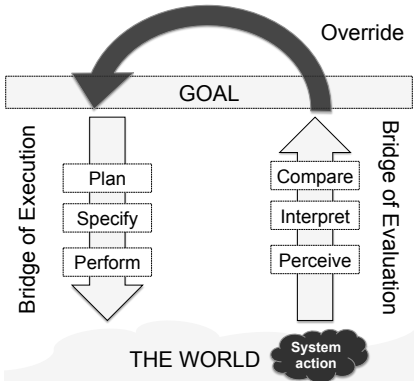


Figure 2.6: When users decide that the system's action is undesired, and wish to override it, we move back to the left side of the action cycle: the execution stage.

of how a context-aware system is acting based on the context, is part of the notion of intelligibility (Bellotti and Edwards, 2001). However, Bellotti and Edwards also discuss that systems should provide more detailed information by informing users about their understanding of the world, and how they have arrived at those conclusions. We come back to these issues when we discuss how we can allow users to cope with the complexity of context-aware systems.

2.3.3 *Coping with the Complexity and Dynamic Behaviour of Context-Aware Systems*

Two characteristics of context-aware systems that complicate interaction, are their complexity and dynamic behaviour (see Section 2.2.1). Under the hood, context-aware systems may use a large set of interconnected rules or machine learning algorithms, that can be hard to understand for end-users (Dey and Newberger, 2009). Users should be able to build up a model of how the system works without being overwhelmed with technical details or possible contradictory information (Norman, 2013b, pg. 183). Moreover, the dynamic behaviour of context-aware systems, where the same input from the user might effectively yield another result depending on the context, makes it hard to predict how the system will behave. Norman (2009, pg. 52) mentions that machines getting more powerful and autonomous opens up a *supergulf* that separates machines and humans, caused by a lack of common ground in communication (Clark and Brennan, 1991). Essentially, what Norman refers to is that on the one hand, people are experiencing difficulties understanding machines and why they act in a certain way, and on the other hand machines will never have a complete picture of our intentions or goals (as also mentioned earlier, e.g., Dourish, 2004). Given the fact that machines cannot communicate and understand the same way we do, Norman (2013b, pg. 67) states that designers "have a special obligation to ensure that the behaviour of machines is understandable to the people who interact with them".

Norman (2013b, pg. 25) suggests, as one of his fundamental design principles, that systems should provide users with a simple conceptual model: "an explanation, usually highly simplified, of how something works". A conceptual model can be incomplete or even inaccurate, but what matters is that it helps users in interacting with a system. Conceptual models help users to predict how the system will behave, and to figure out what to do when things go wrong. However, this is not an easy thing to do, given the complexity and dynamic behaviour that context-aware systems tend to exhibit, especially if we consider context-aware environments consisting of several interconnected systems. Norman (2013b) discusses the Nest thermostat as an example of a system that constantly explains what it is doing, and thus provides continual feedback. He nevertheless notes that the Nest is not perfect and this has also been found in recent studies suggesting that there are still important problems with respect to intelligibility and control (Yang and Newman, 2013), as previously discussed in Section 2.2.2.

Specific stages can be identified within the Stages of Action model where the system can help users build up a conceptual model of its behaviour (Figure 2.7). First, in the stages of evaluation, systems could not only show that they are performing an action, but also explain the reasoning behind that action, which could help users predict the system's future behaviour (as with the previous example of the Nest). Providing this information is what Bellotti and Edwards (2001) refer to as "informing users of the current contextual system's understandings and capabilities". It is important to note that users might also require information from the system when the system did not act, even though they expected it to do so. Secondly, when users are in the execution phase, the system could allow them to explore the consequences of their actions, by giving them information about what would happen when they perform a certain action.

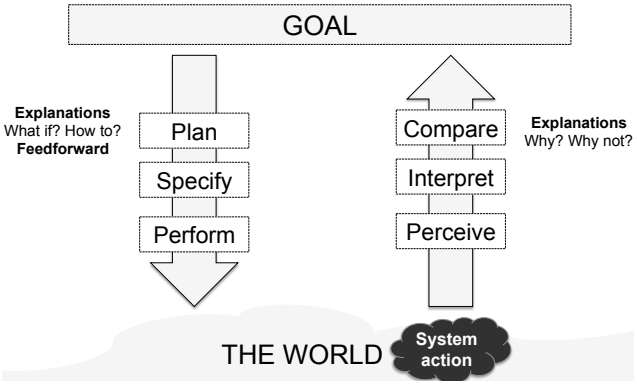


Figure 2.7: Explanations can help users to build up a conceptual model, both in the evaluation and execution stages. Feedforward is useful in the execution phase: it helps users predict what the result of their actions will be.

A common method to explain the system's reasoning is through the use of automatically generated explanations (Cheverst et al., 2005; Lim et al., 2009; Lim and

Dey, 2010; Kulesza et al., 2009; Vermeulen et al., 2009b, 2010). In Chapter 7, we present our work on automatically generated explanations that answer 'why?' and 'why not?' questions about system actions that allow users to build up a conceptual model of the behaviour of a rule-based context-aware system. Lim and Dey (2010) have also explored the use of 'what if?' and 'how to?' questions, which are useful in the stages of execution. Furthermore, in Chapter 5, we introduce another way to help users bridge the gulf of execution: the design principle *feedforward* (Vermeulen et al., 2013b) that designers can employ to show users what the result of their action will be. Feedforward is also mentioned by Bellotti and Edwards (2001) as a specific type of intelligibility information that helps users to predict the consequences of their actions. When the system provides intelligibility during the evaluation phase that helps users build up a conceptual model, users can also use this knowledge in the execution phase to figure out what actions are necessary to achieve their goals.

2.3.4 *Dealing with Implicit Input*

The fact that context-aware systems typically rely on implicit interaction can also cause interaction problems. We already discussed the problems on the evaluation side in our discussion on interacting with autonomous systems (Section 2.3.2). On the execution side, we mainly focus on issues with implicit input.

One of the most important problems caused by the reliance on implicit input is the lack of *visibility* and *discoverability* (Norman, 2013b). Visibility allows users to answer important questions such as 'what is supported by the system?' and 'what can I do?'. Unlike traditional GUI/WIMP systems (see Section 2.2.1), users have no way of exploring the system's functionality. Indeed, this even seems to be implied by the premise of the 'invisible computer', where sensors and components that are hardly noticeable still play a role in performing part of the interaction (e.g., a camera that detects movements in front of a display). Moreover, how do users determine where and how they can provide input to the system? Many systems relying on implicit interaction make it difficult for users to discover how to do this.

We know analyse implicit input in the context of the action cycle. First, consider the situation where the user did not intend to provide input to the system and is surprised when the system suddenly reacts. This is essentially the situation portrayed in Figure 2.5. Although the system is responding to the user's (implicit) input here, unlike when it acts autonomously, the user did not perform any conscious action with a goal in mind. In the user's mind, the system just suddenly acts, not realizing that it actually responded to their actions. It is important in this situation to make sure that users are aware of the system's action (see Section 2.3.2), and to inform users that they caused that action to occur (e.g., see Section 2.3.3). This would improve discoverability and allow users to build up a conceptual model of the system's behaviour by connecting their own actions to the system's responses.

Secondly, consider the situation where a user has formulated a goal, wants to interact with the system, but does not know how to interact with the system. In this case, the user is uncertain about the action possibilities and about how to perform the necessary actions to achieve their goals. As shown in Figure 2.8, this problem

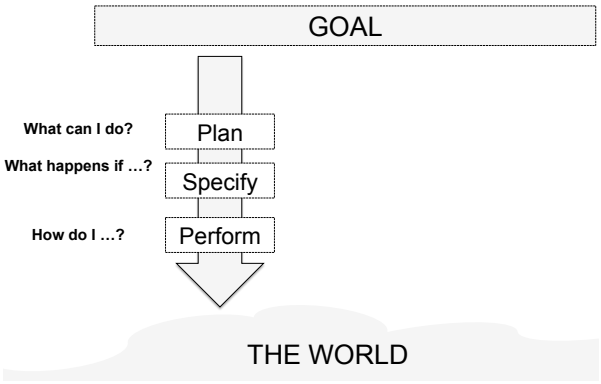


Figure 2.8: Systems that employ implicit interaction can increase the gulf of execution due to a lack of discoverability and visibility of action possibilities.

increases the gulf of execution. Ju's implicit interaction framework (Ju and Leifer, 2008; Ju et al., 2008; Ju, 2008) mentions a number of strategies to improve discoverability, such as *offers* that convey potential actions to users at the right time, and feedforward that tells users what the consequences of their actions are.

In Chapter 8, we investigate how we can convey action possibilities and provide tracking feedback for systems that rely on implicit interaction. More specifically, we explore techniques to improve discoverability and visibility for a proxemics-aware, large public display. Additionally, the design principle feedforward (Chapter 5) can be used to help users in performing the appropriate actions.

2.3.5 *Mapping the Dissertation Chapters to the Seven Stages of Action Model*

This dissertation proposes several techniques to improve intelligibility and control. Figure 2.9 shows how the different techniques (and corresponding chapters) can be situated within Norman's Seven Stages of Action Model. In Chapter 5, we discuss how we can provide users with information about what will happen, allowing them to better bridge the gulf of execution. In Chapter 6, we discuss how systems can make their actions more visible to users, while also offering users sufficient time to intervene. Next, In Chapter 7, we present the use of explanations to provide users with detailed knowledge of the system's decisions, which improves predictability and helps users construct an accurate conceptual model. Finally, in Chapter 8, we explore how we can address interaction challenges in proxemic-aware large interactive surfaces by showing subtle cues on a secondary floor display. This technique mainly addresses problems of feedback (gulf of evaluation), visibility and discoverability (gulf of execution).

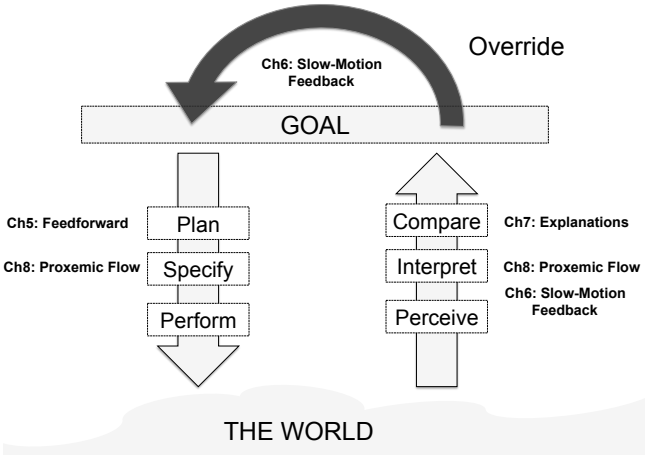


Figure 2.9: Overview of how the different techniques proposed in this dissertation can be situated as design solutions in Norman's Seven Stages of Action.

2.4 CONCLUSION

In this chapter, we introduced the research areas of ubiquitous computing and context-aware computing, in which we situate this dissertation. In addition, we investigated the motivation for this work by surveying interaction challenges with context-aware computing. Based on the existing literature, we then proposed the need for intelligibility and control as a means to address these problems. We further clarified how intelligibility and control can improve interaction with context-aware systems by analyzing those interaction challenges within Norman's Seven Stages of Action.

In chapters 5–8, we delve deeper into the proposed techniques as illustrated in Figure 2.9. Before moving on to these techniques, however, we present a design space that introduces different design choices that play a role when providing intelligibility and control (Chapter 3). In Chapter 4, we present an exploratory study of a context-aware guidance system for nurses to illustrate the different design decisions covered by this design space, and discuss insights gained from that study with respect to the interaction challenges discussed in this chapter.

3

A DESIGN SPACE FOR INTELLIGIBILITY AND CONTROL

3.1 INTRODUCTION

As mentioned in Section 2.2.3, Bellotti and Edwards (2001) state that context-aware systems need to be *intelligible*, and should be able to represent to their users "what they know, how they know it, and what they are doing about it". Moreover, they argue that effective *control* strategies should be provided to allow users to intervene when the system makes a mistake.

Nevertheless, there are many possible ways to design for intelligibility and control, each of which could be suitable in some situations, but not in others. Consider, for example, the subtle cues that Google Maps employs to convey its confidence in the user's location. When the location is accurately tracked, it is visualized on the map using a blue dot. When the application is not entirely sure of the user's location, however, it shows a blue circle around the dot (Figure 3.1). The circle's size indicates the location tracking accuracy: the larger the circle, the larger the area in which the user might be located, and thus also the lower the accuracy. Although this is an effective (and quite natural) way of revealing location tracking inaccuracies, a different intelligibility strategy might be necessary in other circumstances, for example when dealing with non-spatial data or in situations where users are not paying close attention to the display.

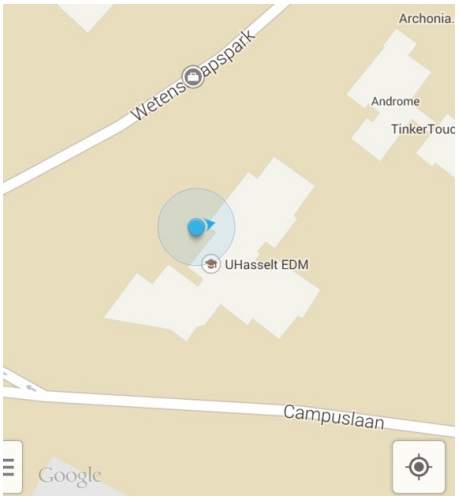


Figure 3.1: Google Maps shows a blue circle that changes in size to convey how confident it is of the user's current location (source: Google Maps for Android, base map © Google Maps 2014).

In this chapter, we explore design decisions that ubicomp designers, developers and researchers face when adding support for intelligibility and control. We start with a literature overview of different techniques that have been employed, after which we introduce a design space.

3.2 TECHNIQUES TO SUPPORT INTELLIGIBILITY AND CONTROL

Several techniques have been explored to support end-users in understanding context-aware systems, and allowing them to control, configure or even program context-aware ubicomp environments. In what follows, we provide a short literature overview.

3.2.1 *Support for Intelligibility: Improving Understanding*

3.2.1.1 *Textual Explanations*

One of the most studied techniques to support intelligibility is the use of automatically generated *textual explanations* (e.g., Cheverst et al., 2005). In particular, explanations answering 'why?' and 'why not?' questions have been investigated intensively (e.g., Lim et al., 2009; Lim and Dey, 2009, 2010; Kulesza et al., 2009; Vermeulen et al., 2010). In Chapter 7, we discuss our own work on why questions with the PervasiveCrystal system (Vermeulen et al., 2009b, 2010).

The use of explanations originated in research on making 'intelligent systems' understandable for end-users, and has been applied, for example, in expert systems (see Gregor and Benbasat, 1999 for an overview) and recommender systems (e.g., Cramer et al., 2008). In addition, explanations have also been investigated for traditional GUI applications. Myers et al. (2006) used automatically generated answers to 'why' and 'why not?' questions in a word processor to help end-users understand complex behaviours and interdependencies among various of the application's features (answering questions such as 'Why is this text bold?'). On-line stores such as Amazon also use why questions in combination with high-level explanations to help shoppers understand 'why' some articles are recommended to them, as shown in Figure 3.2.

3.2.1.2 *Visualizations*

Another common approach to make systems more intelligible is through the use of visualizations. Visualizations have been used to improve understanding of the system's state and behaviour, show action possibilities, or show the outcome of an action.

In fact, the circle visualization shown in Figure 3.1, is an effective way to display one aspect of the state of a location-aware system: its confidence in determining the user's location. Variations of this type of location error visualization have also been explored in the literature (e.g., Dearman et al., 2007; Aksenov et al., 2012; Lemelson et al., 2008; Burigat and Chittaro, 2011; Kjærgaard and Weckemann, 2012). Uncertainty in general is often represented in text: using numerical values (e.g., Lim

Figure 3.2: Explanations for recommendations in Amazon's Kindle Store.

and Dey, 2011a; Antifakos et al., 2005), sometimes combined with colour ranges (e.g., Rukzio et al., 2006; Lemelson et al., 2008), or using categorical values (e.g., Cheverst et al., 2005). In Chapter 8, we discuss visualizations that indicate how well users are tracked in front of a public display and make them aware of potential tracking problems.

Rehman et al. (2005) investigated visualizing interaction zones for a location-aware application. Their application allows users to bring up their personal desktop environment on any nearby display in the office. Wearing a head-mounted display, users see a visualization of the exact zone in which they needed to be in order to trigger the 'desktop teleport' feature. Although this technique is somewhat clunky due to the use of a head-mounted display, the invisibility of interaction zones is indeed a common problem in ubicomp environments, which has also re-emerged in studies of interactive public displays (e.g., Jurmu et al., 2013). In Chapter 8, we discuss our approach to visualize interaction zones for people-aware public displays using a secondary floor display.

Next to textual explanations, we can also use visualizations to make users aware of how the system works, and why it is behaving in a certain way. In Section 6.3, we discuss projected visualizations in a smart room that show events firing in real-time (Vermeulen et al., 2009a). Furthermore, by revealing connections between system actions and the different devices and sensors in the room, we provide users with an understanding of the system's behaviour (i.e., by explaining what caused system actions to be executed). Lim and Dey (2011a) also explored several custom visual explanations to make a social awareness application intelligible. This application used sensors to give users an impression of the availability of others, i.e., whether it

would be appropriate to interrupt them. The explanations were specific to the sensor data and the type of information that was inferred from that data. For instance, a pan flute metaphor was used to represent sound pitch when explaining inferences of whether the person was talking to someone or listening to music. Dey and Newberger (2009) introduced the Situations framework—an extension to the Context Toolkit (Dey et al., 2001)—that allows designers to connect custom-designed interfaces (and visualizations) to the internal logic of context-aware applications. One of their example applications shows a unified interface to monitor and control context-aware lighting and temperature in a smart home. Similarly, the Nest smart thermostat can be monitored and controlled through mobile and web applications that allow users to consult visualizations of its current state, energy consumption and schedule (Rogers, 2013).

Visualizations can also help users understand the outcomes of their own actions. The design principle *feedforward* (see Chapter 5) tells users what the result of an action will be (Vermeulen et al., 2013b; Bellotti and Edwards, 2001). In Section 5.6, we discuss the Feedforward Torch, a combination of a smartphone and mobile projector that projects visualizations on objects in users' everyday environments (such as light switches or complex appliances). The Feedforward Torch shows feedforward to inform users of the consequences of their own actions. Visual animations of action outcomes were especially deemed useful when the action outcome either did not happen immediately, or when it would be invisible to the user (e.g., a light switch that turns on the lights upstairs). Visualizations of the outcomes of users' actions have also been employed in the OctoPocus gestural guide (Bau and Mackay, 2008). Just like many context-aware systems, gestural interaction lacks proper visibility of action possibilities, which makes it hard for users to know which gestures are available and what commands they trigger. OctoPocus helps users perform gestures by continuously showing the possible remaining gesture paths, and what commands are triggered by those gestures.

Finally, visualizations have been used to improve awareness of system actions in context-aware systems. In their proximity-aware interactive whiteboard, Ju et al. (2008) use animations to show the outcome of system actions. For example, when the user is approaching the whiteboard, it switches from ambient mode to drawing mode, which means that all content will be cleared. It makes this switch apparent by animating each content element on a path from the centre of the whiteboard to the side. Additionally, the whiteboard visualizes how it has interpreted the user's input. When users step back, the whiteboard shows outlines around the users' ink strokes to show how they are being clustered. In Chapter 6, we introduce a general technique to improve awareness of system actions: *slow-motion feedback* (Vermeulen et al., 2014).

3.2.2 *Support for Control: Allowing Users to Intervene*

Several approaches to support end-users in controlling, configuring or programming ubicomp environments have been explored in the literature. We focus our

overview mainly on tools and techniques that extend control to end-users, and thus exclude rapid prototyping tools targeted at developers.

3.2.2.1 *Overriding System Actions*

One of the simplest control mechanisms is allowing users to *override* actions performed by the system, similar to an 'undo' command. Note that in this case, however, the user is not undoing an action they did before, but an action performed by the system. For example, when content moves from the centre of Ju et al.'s interactive whiteboard to the side to make space, users can simply grab the content to stop it from moving. We also provide users with a *cancel* command when projecting visualizations (see Section 6.3), and when showing answers to 'why questions' (see Chapter 7). A related technique is to allow users *mediate* or correct the system's inferences (Dey and Mankoff, 2005). Mediation techniques allow the user to go into dialogue with the system to fix incorrect inferences and teach the system to avoid making similar mistakes in the future.

3.2.2.2 *Configuration Interfaces*

Another common approach is providing users with the means to configure the system's state and part of its behaviour using specific configuration interfaces. Cheverst et al. (2005) shows visualizations of decision tree rules in their IOS system and allows end-users to manipulate system parameters. As mentioned earlier, Situations (Dey and Newberger, 2009) supports custom intelligibility and control interfaces that can be connected to the underlying logic and parameters of a context-aware application. Dey and Newberger (2009) discuss an example that allows a museum administrator to configure how context-aware displays near exhibits react to people's presence. Similarly, in our own work on explanations (see Chapter 7), we also allow users to bring up a specific configuration interface. For example, when users ask why the lights went out, they can either override that system action (and turn the lights on again), or bring up a specific control interface that allows them to configure specific details such as the light's intensity, or to select another light to turn on instead (Vermeulen et al., 2010).

3.2.2.3 *End-User Programming*

Finally, the most powerful kinds of techniques—but also the most difficult ones to make accessible to end-users—provide the means to (re-)program ubicomp environments. For example, iCAP (Dey et al., 2006) is a tool that allows users to prototype context-aware applications without having to write code. Observations from a study revealed that end-users without a programming background were able to use iCAP to create reasonably complex applications. With iCAP, users build up rules that associate 'situations' to actions, using a *visual rule building interface*, that they can afterwards try out using a simulator. Similarly, Rodden et al. (2004) developed a tool to construct rules using the metaphor of assembling jigsaw pieces, representing sensors or operations. Others have looked at more physical ways to construct

rules and configure system behaviour. Kawsar et al. (2008) allowed end-users to deploy context-aware services in their homes by linking applications and artefacts—augmented objects in the home, such as a toothbrush with integrated accelerometer or a mirror with integrated display—together using RFID cards.

The previous tools, however, all rely on a rule-based specification of behaviour. Alternatively, recognition-based approaches that rely on machine learning might be necessary to reliably sense certain aspects of context. For example, machine learning might be necessary to recognize patterns in accelerometer data or to analyse ambient acoustics to determine whether someone is talking. Although these could also be supported in a rule-based approach as specific recognition components, this would prevent users from creating their own recognizers. Dey et al. (2004) developed 'a CAPpella', a tool that involves end-users in specifying these rules by allowing them to *demonstrate* the desired behaviour to the system, after which a machine-learning algorithm tries to generalize the user's inputs—also known as *programming by demonstration* (Cypher et al., 1993). Additionally, researchers have been investigating whether users can understand and modify machine-learned systems (Kulesza et al., 2009, 2013).

3.3 DESIGN SPACE

The techniques that were discussed in the previous section typically only represent a single point in the larger *design space* of possible strategies for intelligibility and control. In order to compare different techniques and generate design alternatives, we introduce a design space for intelligibility and control techniques (Vermeulen, 2010; Vermeulen et al., 2013a) consisting of six dimensions (Figure 3.3).

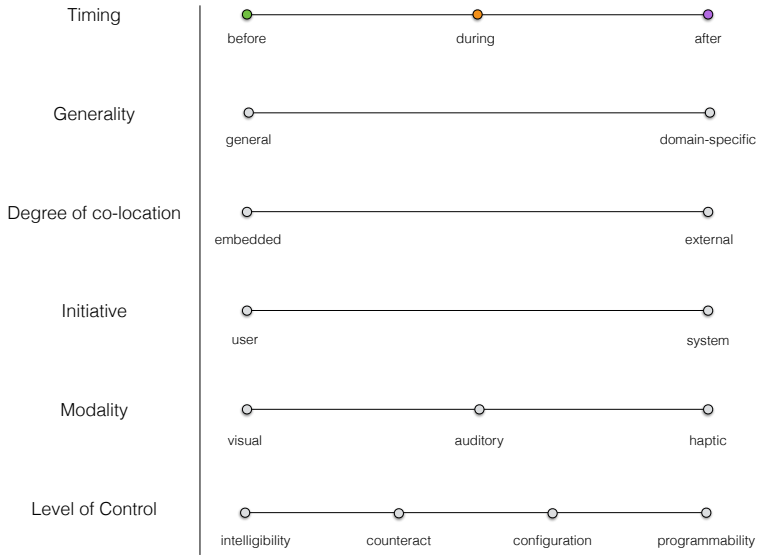


Figure 3.3: The design space for intelligibility and control, consisting of six dimensions.

A design space presents both the design choices that need to be made, together with the alternatives for these decisions (Shaw, 2012). The dimensions of the design space represent the design decisions, while the values on those dimensions represent possible alternatives. Design space analysis is a common technique in HCI, with the aim of comprehending different designs as points in a multi-dimensional space formed by the design space dimensions (e.g., Card et al., 1991; Fitzmaurice et al., 1995; Coutaz, 2007).

The six dimensions of the design space in Figure 3.3 were derived from a review of existing techniques for intelligibility and control, and a meta-analysis of existing taxonomies and frameworks with respect to interaction challenges in context-aware and ubiquitous computing. Additionally, our design space was informed by insights on the impact of different design choices from designing a context-aware guidance system for nurses, as discussed later in Chapter 4.

We start with a brief overview of the design space dimensions, after which we discuss each of them in more detail and position different techniques in the design space:

- *Timing*: Both intelligibility and control can be supported at certain times during the interaction: *before*, *during* and *after* events take place.
- *Generality*: User interfaces and interaction techniques for intelligibility and control can be *general* (i.e., when they can be applied across different types of applications and domains) or *domain-specific*.
- *Degree of co-location*: Support for intelligibility or control might be *embedded*, or integrated with the rest of the user interface, versus *external*, when users need to switch to a separate interface.
- *Initiative*: Users may have to explicitly request intelligibility information or means to control the system (*user* initiative), or might automatically be presented with these features when necessary (*system* initiative).
- *Modality*: Several modalities can be used to help users to understand or control the system (e.g. *visual*, *auditory*, *haptic*).
- *Level of Control*: The level of control end-users can exert over the system ranges from *intelligibility*, where no additional control is added beyond intelligibility, over *counteract*, where users can perform the opposite action, to *configuration*, where users can tweak predefined system parameters, and finally *programmability* where users can themselves (re-)define how the system works.

Even though this design space for intelligibility and control is non-exhaustive, we believe it is valuable as an analytical tool and can play a generative role to help designers come up with alternative designs. In the remainder of this chapter, we will represent the design space of Figure 3.3 using a table for brevity and clarity.

3.3.1 *Timing*

Intelligibility and control can be provided at different phases during the interaction. We differentiate between different moments in time relative to when events take place: *before*, *during* or *after* that event. We define an event as either an action initiated by the user or an action initiated by the system.

For example, consider the case where the system acts autonomously by responding to the context (see Section 2.3.2 and Figure 2.5):

- *Before* the system action, the system could inform the user of what it is going to do (*intelligibility*), and for example, allow the user to reject or accept that action (*control*).
- *During* the action, the system could slow down the action to allow users to notice what is happening (see Chapter 6), or help users to construct a conceptual model of the system's behaviour by visualizing the interplay between triggers and consequences of the action (see Section 6.3). Both examples are strategies to support *intelligibility*. Systems can provide *control* at this stage by allowing users to override actions that are in progress.
- *After* the action, the system could allow the user to request detailed information about that action or what caused it, e.g., by providing explanations (*intelligibility*), revert the system to its previous state if the action was undesired (*control*), or provide a historical trace of past events (*intelligibility*).

In Table 3.1, we show how different techniques can be viewed with respect to the timing dimension. The last four rows represent our own systems and techniques, which we will present in detail in Chapters 5–8.

Lim and Dey's concept of 'what if?' questions (Lim and Dey, 2009, 2010) is an example of intelligibility that is provided before any action (by the system or user) is performed. They allow the user to know what an application would do given a set of inputs that characterize a certain situation (or context). Also, the interaction zone visualization by Rehman et al. (2005) is an example of information provided before the action; it helps users know what actions are possible, and how to perform an action (i.e., in what region they need to be).

Ju et al.'s proximity-aware whiteboard (Ju et al., 2008) can be seen as providing intelligibility and control before, but mostly during system actions. As discussed before, with their 'system demonstration' technique, the system shows the user what it is doing or going to do, such as switching from ambient mode to drawing mode upon approach and clearing the existing content. Moreover, while the whiteboard is transitioning between modes, users can grab the moving contents to cancel the mode switch (a technique Ju et al. refer to as 'override'). Similarly, while users are performing gestures, gesture guides such as OctoPocus (Bau and Mackay, 2008) or ShadowGuides (Freeman et al., 2009) show the possible gestures and feedforward about the commands associated with those gestures. The principle of *just-in-time chrome* (Wigdor and Wixon, 2011, pg. 152) where gestures are made self-revealing

Timing

	Before	*During*	*After*
Amazon's explanations			✔
Crystal (Myers et al., 2006)			✔
IOS (Cheverst et al., 2005)		✔	✔
Google Maps location error		✔	
Range (Ju et al., 2008)	✓	✔	
Visualizing interaction zones (Rehman et al., 2005)	✔	✓	
"Just-in-time chrome" (Wigdor and Wixon, 2011)		✔	
OctoPocus (Bau and Mackay, 2008)	✓	✔	
ShadowGuides (Freeman et al., 2009)	✓	✔	
TouchGhosts (Vanacken et al., 2008)	✓	✔	
Situations (Dey and Newberger, 2009)		✔	✔
"What if?" (Lim and Dey, 2009, 2010)	✔		
Feedforward Torch (Vermeulen et al., 2012)	✔		
PervasiveCrystal (Vermeulen et al., 2010)			✔
Visible Computer + Slow-Motion Feedback (Vermeulen et al., 2009, 2014)		✔	
Proxemic Flow (Vermeulen et al., 2014)	✔	✔	✔

Table 3.1: The timing dimension allows us to distinguish between techniques that provide information before, during, or after the action. Note that coloured boxes with check marks indicate a technique's primary classification, while small check marks indicate a possible alternative classification.

with 'just-in-time' extra on-screen graphics is another example of supporting intelligibility during the action.

Finally, answers to 'why?' and 'why not?' questions (e.g., Myers et al., 2006; Kulesza et al., 2009; Lim and Dey, 2010; Vermeulen et al., 2010; see Chapter 7), explanations for recommender systems (e.g., as in Amazon's store, see Figure 3.2) and Cheverst et al.'s explanations are examples of intelligibility information provided after the action. A means to 'undo' the action performed by the system, is an example of control provided after the action.

3.3.2 *Generality*

Intelligibility or control techniques can be generally applicable (*general*) or specific to a certain domain or a specific type of application (*domain-specific*), as illustrated in Table 3.2.

An example of a domain-specific intelligibility interface is the location tracking accuracy visualization used in Google Maps, as discussed before (see Figure 3.1). While domain-specific techniques might limit flexibility and reuse, they might be easier for users to understand as they can be more easily expressed in terms of the user's goals (i.e., they provide a better match). After all, it is easier to esti-

Generality

	General	*Domain-specific*
Amazon's explanations	✔	✔
Crystal (Myers et al., 2006)	✔	✔
IOS (Cheverst et al., 2005)	✔	
Google Maps location error		✔
Range (Ju et al., 2008)	✔	✔
Visualizing interaction zones (Rehman et al., 2005)		✔
"Just-in-time chrome" (Wigdor and Wixon, 2011)		✔
OctoPocus (Bau and Mackay, 2008)		✔
ShadowGuides (Freeman et al., 2009)		✔
TouchGhosts (Vanacken et al., 2008)		✔
Situations (Dey and Newberger, 2009)	✔	✔
"What if?" (Lim and Dey, 2009, 2010)	✔	
a CAPpella (Dey et al., 2004)	✔	
iCAP (Dey et al., 2006)	✔	
Jigsaw editor (Rodden et al., 2004)		✔
Feedforward Torch (Vermeulen et al., 2012)	✔	✔
PervasiveCrystal (Vermeulen et al., 2010)	✔	
Visible Computer + Slow-Motion Feedback (Vermeulen et al., 2009, 2014)	✔	
Proxemic Flow (Vermeulen et al., 2014)		✔

Table 3.2: The generality dimensions differentiates between techniques that are generally applicable versus domain- or application-specific ones.

mate the impact of imprecise location tracking by interpreting a visualization of the possible error on a map than by interpreting an error percentage. In addition, domain-specific techniques are typically easy to integrate into applications for that same domain (see also the co-location dimension: Section 3.3.3), which can help users to stay concentrated on their current task. Our Proxemic Flow techniques for people-aware public displays (see Chapter 8) are also domain-specific. An example of a general interface for intelligibility and control is PervasiveCrystal (Vermeulen et al., 2010) (see Chapter 7), which provides users with the possibility to pose why and why not questions about any event occurring in a ubicomp environment and also offers simple control primitives.

Nevertheless, it might sometimes be difficult to categorize a technique as general or domain-specific. For example, even though Ju et al.'s *user reflection*, *system demonstration* and *override* techniques are generally applicable, their implementation of those techniques for the Range whiteboard is fairly specific to that type of application.

3.3.3 *Co-location*

The co-location dimension refers to the level of integration between an interface for intelligibility and control, and the application in which it is being employed. Intelligibility or control could be offered in a separate interface (*external*), or could be an integrated part of the application (*embedded*). In her discussion on meta-user interfaces (user interfaces that support *evaluating* and *controlling* the state of a smart space), Coutaz (2007) makes a similar distinction; she refers to this dimension as the 'level of integration'.

Degree of co-location

	Embedded	*External*
Amazon's explanations	✔	
Crystal (Myers et al., 2006)	✔	
IOS (Cheverst et al., 2005)		✔
Google Maps location error	✔	
Range (Ju et al., 2008)	✔	
Visualizing interaction zones (Rehman et al., 2005)	✔	
"Just-in-time chrome" (Wigdor and Wixon, 2011)	✔	
OctoPocus (Bau and Mackay, 2008)	✔	
ShadowGuides (Freeman et al., 2009)	✔	
TouchGhosts (Vanacken et al., 2008)	✔	
Situations (Dey and Newberger, 2009)	✔	✔
"What if?" (Lim and Dey, 2009, 2010)	✔	✔
a CAPpella (Dey et al., 2004)		✔
iCAP (Dey et al., 2006)		✔
Jigsaw editor (Rodden et al., 2004)		✔
Feedforward Torch (Vermeulen et al., 2012)	✔	✔
PervasiveCrystal (Vermeulen et al., 2010)		✔
Visible Computer + Slow-Motion Feedback (Vermeulen et al., 2009, 2014)	✔	
Proxemic Flow (Vermeulen et al., 2014)	✔	

Table 3.3: The degree of co-location dimension allows us to separate techniques that are are embedded in the application from techniques that are used through an external interface.

The idea of an embedded interface for intelligibility is somewhat similar to the general notion of embedded help (Nielsen, 1993). Techniques that help users perform gestures such as OctoPocus (Bau and Mackay, 2008), TouchGhosts (Vanacken et al., 2008) or ShadowGuides (Freeman et al., 2009) are therefore also typically embedded (see Table 3.3). Indeed, Freeman et al. (2009) also classify gesture learning systems based on the "degree of co-location of the learning space and performance space". Their notion of 'in-situ' gesture learning systems, where learning gestures is integrated into the same mode where gestures are performed (i.e., the user's

current task), is similar to our concept of an embedded interface for intelligibility. Likewise, the principle of *just-in-time chrome* (Wigdor and Wixon, 2011) to provide self-revealing gestures with a minimum of extra on-screen graphics is another embedded technique.

Most domain-specific techniques tend to be embedded within the user's current application. For example, Ju et al.'s techniques (Ju et al., 2008) and the Google Maps location error visualization are additional examples of embedded intelligibility interfaces. In their observations of how people use the Nest thermostat, Yang and Newman (2013) argue for providing intelligibility "opportunistically and in small pieces", corresponding to the occasional, incidental interactions that users have with the Nest. They call this *incidental intelligibility*, or "interaction elements that increase users' understanding of the system's intelligent behaviour *embedded* in the tasks they consciously seek to accomplish" [emphasis ours] (Yang and Newman, 2013), which is again analogous to the idea of embedded intelligibility.

External interfaces tend to be useful for controlling or understanding high-level, generic components of a system. Examples of external interfaces are iCAP (Dey et al., 2006) and a CAPpella (Dey et al., 2004). While users only need to learn how to use external interfaces once, unlike embedded interfaces, they might require the user to interrupt their task and switch to a separate mode.

In ubicomp spaces, instead of an application running on a machine, the 'interface' in which intelligibility is to be integrated, might be the user's environment itself (see also Nielsen's argument about the shift in interface locus). Our original definition of co-location still applies: co-location then becomes the level of integration with the *environment*. Because of the heterogeneous nature of ubicomp spaces, co-located information can indeed be very useful. In an experiment with Pervasive-Crystal (Vermeulen et al., 2010) (another *external* interface), we observed that users occasionally felt disconnected from the explanations we provided (e.g., "the lights turned on because motion was detected"). They were, for example, confused over where in the room motion had been detected by the system (see Chapter 7). Intelligibility and control interfaces that are embedded into the environment can direct users attention appropriately to a specific point of interest. For example, our 'Visible Computer' technique where we show projected event visualizations (Vermeulen et al., 2009a) (see Section 6.3), informs users not only about what is is happening, but also *where* it is happening. Similarly, the Feedforward Torch (Vermeulen et al., 2012b) (Section 5.6), our Proxemic Flow techniques (Chapter 8) and the augmented reality system by Rehman et al. (2005) show co-located guidance.

3.3.4 *Initiative*

The initiative for showing information to improve the users' understanding can be taken by the system itself (*system* initiative) or this information can be made available upon request by the user (*user* initiative). When the system takes the initiative, it could reveal information to draw the user's attention to a certain event, as with Ju et al.'s system demonstration technique. Alternatively, the system could provide users with the means to request detailed information if they need it, similar to the

way services like Amazon allow users to ask why certain products were recommended to them (see Figure 3.2). In Table 3.4, we categorize different techniques with respect to who takes the initiative.

	Initiative	
	User	*System*
Amazon's explanations	✔	
Crystal (Myers et al., 2006)	✔	
IOS (Cheverst et al., 2005)	✔	
Google Maps location error		✔
Range (Ju et al., 2008)		✔
Visualizing interaction zones (Rehman et al., 2005)		✔
"Just-in-time chrome" (Wigdor and Wixon, 2011)		✔
OctoPocus (Bau and Mackay, 2008)	✔	✔
ShadowGuides (Freeman et al., 2009)	✔	✔
TouchGhosts (Vanacken et al., 2008)	✔	✔
Situations (Dey and Newberger, 2009)	✔	
"What if?" (Lim and Dey, 2009, 2010)	✔	
Feedforward Torch (Vermeulen et al., 2012)	✔	
PervasiveCrystal (Vermeulen et al., 2010)	✔	
Visible Computer + Slow-Motion Feedback (Vermeulen et al., 2009, 2014)		✔
Proxemic Flow (Vermeulen et al., 2014)	✔	✔

Table 3.4: The initiative dimension represents whether information is is available upon request, or rather whether it is provided automatically when deemed necessary.

The Feedforward Torch (Section 5.6) is an example of a user-driven technique. Users point the device at an object in their environment (e.g., a light switch), after which it projects feedforward information related to that object. The PervasiveCrystal system where we allow users to pose why questions is also user-driven. In contrast, our 'Visible Computer' technique where we project the event flow between different devices and sensors (Vermeulen et al., 2009a) is a system-driven technique.

There might be several arguments for choosing between these two strategies. Automatically providing information all the time might be distracting or even annoying for the user. We observed that with the Visible Computer, some users mentioned that they sometimes received too much information, which confused them (see Section 6.3 for more details). When automatically providing information, it is tricky to find the right balance between the amount of detail that is provided, to not overwhelm users but still make sure the provided information is sufficiently detailed to be useful. In select cases, it can be useful to have access to very detailed information to debug the system's behaviour and understand deeper details of how the system works. In this case, we would recommend to leave the initiative of showing this information to the user, so that the information is only there when necessary. On

the other hand, simple and informative feedback that explains to users what the system is doing might be useful to show at all times, even for expert users.

One way to make sure that the user is not overloaded with information when relying on system initiative, is to distinguish between novices and experts. To illustrate this, we again draw on the domain of gestural interaction. An elegant way of supporting the transition from novice to expert, can be found in marking menus (Kurtenbach et al., 1993). Marking menus are a variant of pie menus that support two types of interaction: either popping up the circular menu and selecting an item by tapping, or making a straight 'mark' (or gesture) in the direction of the desired menu item without revealing the menu. The menu only pops up after a certain time (commonly called the 'dwell time'), which naturally occurs when users hesitate. Experts who already know which menu item they want to select, can make a mark in that direction instead. In this way, the system has a way to distinguish between novices and experts, and can thus only show the pie menu when necessary. Moreover, Wigdor and Wixon (2011) note that the software for marking menus is identical in novice and expert mode: "the user simply uses the system faster and faster". Because there is really almost no change in interaction, users do not experience a drop in speed of performance when switching from novice to expert mode (what Wigdor and Wixon refer to as 'The Gulf of Competence'). Note that when an expert is unsure, they can still slow down to see the pop-up menu. In that sense, this kind of technique can also be seen as a mixed approach, as users can still deliberatively trigger it (note the small check marks in the 'User' column in Table 3.4). Advantages of this method include this smooth transition from novice to expert (and back, if necessary), and the fact that experts are not slowed down by the interface for novice users. A similar approach of reacting to the user's hesitation is used in OctoPocus (Bau and Mackay, 2008). In our Proxemic Flow techniques (Chapter 8), we also show guidance after a certain dwell time. Nevertheless, the general problem of distinguishing between novices and experts is not an easy one to solve. One could even argue that the generalized problem is a *meta-problem*, where we would need perfect context sensing to tailor intelligibility information to the user (i.e., the *context* that we would need to sense here consists of the user's previous experience and skills)—which we know is close to impossible (see Section 2.2.2).

3.3.5 *Modality*

Depending on the domain and the context of use, some modalities might be more appropriate than others. When driving a vehicle, for example, users need to keep their eyes on the road. This makes interfaces that require visual attention or accurate pointing inappropriate, making it necessary to provide intelligibility or control using other modalities. For this dimension, we distinguish between *visual*, *haptic* and *auditory* interaction. It is important to note that these modalities are not mutually exclusive. Systems can also support intelligibility using multiple modalities (e.g., providing both visual and auditory information), which may or may not be used simultaneously during the interaction.

Modality

	Visual	*Auditory*	*Haptic*
Amazon's explanations	✔		
Crystal (Myers et al., 2006)	✔		
IOS (Cheverst et al., 2005)	✔		
Google Maps location error	✔		
Range (Ju et al., 2008)	✔		
Visualizing interaction zones (Rehman et al., 2005)	✔		
"Just-in-time chrome" (Wigdor and Wixon, 2011)	✔		
OctoPocus (Bau and Mackay, 2008)	✔		
ShadowGuides (Freeman et al., 2009)	✔		
TouchGhosts (Vanacken et al., 2008)	✔		
Situations (Dey and Newberger, 2009)	✔		
"What if?" (Lim and Dey, 2009, 2010)	✔		
a CAPpella (Dey et al., 2004)	✔		
iCAP (Dey et al., 2006)	✔		
Jigsaw editor (Rodden et al., 2004)	✔		
Feedforward Torch (Vermeulen et al., 2012)	✔		✔
PervasiveCrystal (Vermeulen et al., 2010)	✔		
Visible Computer + Slow-Motion Feedback (Vermeulen et al., 2009, 2014)	✔	✔	
Proxemic Flow (Vermeulen et al., 2014)	✔		

Table 3.5: The modality dimension indicates what modality is used to convey information or exert control over the system.

Most systems typically only support intelligibility or control using the *visual* modality, as is apparent from Table 3.5. In our own work, we have also mostly focused on visual information, bar a few exceptions. For the Visible Computer, we provided functionality to override the system by uttering the speech command 'cancel'. Nevertheless, we observed that speech input is accompanied by its own problems, such as the lack of discoverability (see Section 6.3). With the Feedforward Torch, we also experimented with vibration to notify the user of when information was available. Our slow-motion feedback technique (see Chapter 6) can be used with several modalities, but our application of the technique in the Visible Computer (Vermeulen et al., 2009a) only relies on visual information.

Within the visual modality, there are of course also possible variations, such as textual information versus visuals or animations. Studies have found that compared to written instructions, animated visual instructions might increase performance and reduce mistakes (Palmiter and Elkerton, 1993) and might be preferred for tasks dealing with human movement (Wong et al., 2009). Nevertheless, there are also some disadvantages to animations: there is less retention over time compared to textual instructions (Palmiter and Elkerton, 1993) and motion in animations is often misinterpreted (Tversky et al., 2002). In our own experiments with the Feedforward Torch

(Vermeulen et al., 2012b), participants preferred visualizations and animations over written instructions, which they found more time-consuming to interpret (see Section 5.6). Moreover, animations were appreciated for situations where the action outcome was not immediately perceivable.

3.3.6 *Level of Control*

We distinguish between four increasing levels of control that end-users can exert over a system. Table 3.6 classifies the different techniques based on the level of control they support.

Level of control

	Intelligibility	*Counteract*	*Configuration*	*Programmability*
Amazon's explanations	✔			
Crystal (Myers et al., 2006)	✔	✔		
IOS (Cheverst et al., 2005)	✔	✔	✔	✔
Google Maps location error	✔			
Range (Ju et al., 2008)	✔			
Visualizing interaction zones (Rehman et al., 2005)	✔			
"Just-in-time chrome" (Wigdor and Wixon, 2011)	✔			
OctoPocus (Bau and Mackay, 2008)	✔			
ShadowGuides (Freeman et al., 2009)	✔			
TouchGhosts (Vanacken et al., 2008)	✔			
Situations (Dey and Newberger, 2009)	✔	✔	✔	
"What if?" (Lim and Dey, 2009, 2010)	✔			
a CAPpella (Dey et al., 2004)				✔
iCAP (Dey et al., 2006)				✔
Jigsaw editor (Rodden et al., 2004)				✔
Feedforward Torch (Vermeulen et al., 2012)	✔			
PervasiveCrystal (Vermeulen et al., 2010)	✔	✔	✔	
Visible Computer + Slow-Motion Feedback (Vermeulen et al., 2009, 2014)	✔	✔		
Proxemic Flow (Vermeulen et al., 2014)	✔			

Table 3.6: The level of control dimension indicates what means users have to control the system.

As discussed before (see Section 2.2.3), *intelligibility* can be viewed as the most basic level of control, where users exert control over a system based on their understanding of how it works. An example of this level of control would be only the availability of explanations about the system's behaviour. Based on the understanding gained from these explanations, users could then alter their behaviour to attempt to control the system. The next level of control is *counteracting*. Next to providing intelligibility, systems that provide this level of control also allow users to override the system's actions. Examples of this level of control are PervasiveCrystal's 'undo' command (Vermeulen et al., 2010), Ju et al.'s 'override' technique or the "Don't use for recommendations" button in Figure 3.2. Next are systems that

allow users to tweak predefined system parameters support *configuration*, such as Dey and Newberger's Situations framework (Dey and Newberger, 2009), or our configuration interface in PervasiveCrystal. The most advanced level of control, *programmability*, is available when users can themselves (re-)define how the system works, such as in iCAP (Dey et al., 2006), a CAPpella (Dey et al., 2004), the Jigsaw editor (Rodden et al., 2004), and Cheverst's IOS system (Cheverst et al., 2005).

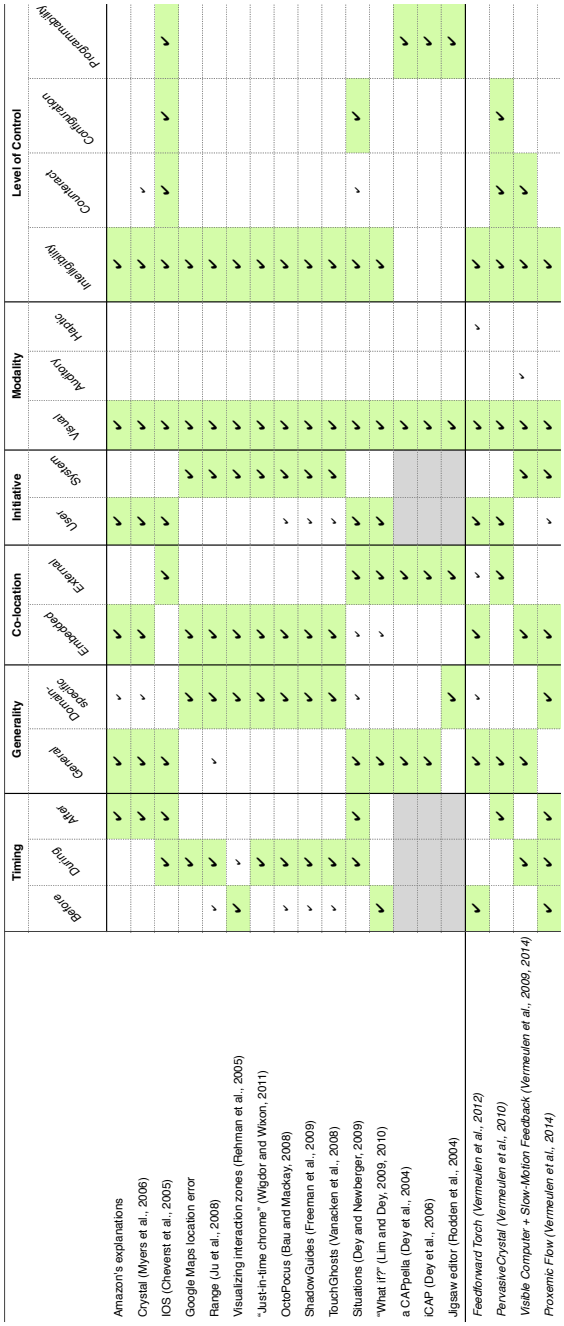
3.4 INSIGHTS FROM MAPPING THE DESIGN SPACE

In Table 3.7, we show an overview table of how the different techniques can be mapped into the design space. Note that grey areas in the table indicate when a certain dimension is not applicable to that specific technique (i.e., initiative and timing make no sense for end-user programming tools). We will now briefly discuss insights for each of the six dimensions, and explore how they can be related to each other.

TIMING This dimension has been mostly underexplored in previous work. There are few general techniques to provide intelligibility and control across the timing dimension, especially before and during actions. Additionally, only a couple of systems provide information about what *will* happen (the *before* column in the table). In our own work, we have focused on this aspect through feedforward (Chapter 5, illustrated with the Feedforward Torch in Section 5.6). Some techniques span multiple alternatives of the timing dimension, such as Ju et al. (2008), the IOS system (Cheverst et al., 2005), and our Proxemic Flow case study (Chapter 8), but most only offer a specific moment in time at which intelligibility is provided. Ideally, systems should be intelligible about past, present and future events.

GENERALITY We covered a variety of domain-specific and general interfaces in our design space. Some systems or techniques are essentially general but can be implemented in a domain-specific way, e.g., recommender systems, what-if questions and Situations (Dey and Newberger, 2009). Most of our own work can be classified as general techniques, although we have also explored domain-specific ones, such as Proxemic Flow.

DEGREE OF CO-LOCATION We notice that most domain-specific interfaces are also *embedded* (e.g., the location error visualization in Google Maps), while most general interfaces are *external*. However, this is not always the case. For example, Rodden's jigsaw editor (Rodden et al., 2004) is a domain-specific interface for controlling a smart home, but is nevertheless external. Moreover, systems that provide intelligibility *during* actions, tend to be embedded. An explanation for this may be that it is easier to provide information during actions when the intelligibility technique is integrated with the user's current task (e.g., as in OctoPocus). In our own work, we have explored both embedded and external techniques, although we believe that embedded techniques show the greatest promise (see also the findings by Yang and Newman, 2013).



	Timing			Generality		Co-location		Initiative		Modality			Level of Control			
	Before	During	After	General	Domain-specific	Embedded	External	User	System	Visual	Auditory	Haptic	Intelligibility	Counteract	Configuration	Programmability
Amazon's explanations	✓		✓	✓	✓	✓		✓		✓			✓			
Crystal (Myers et al., 2006)			✓	✓	✓	✓		✓		✓			✓	✓		
IOS (Cheverst et al., 2005)		✓	✓	✓			✓	✓		✓			✓	✓	✓	✓
Google Maps location error		✓			✓				✓	✓			✓			
Range (Ju et al., 2008)	✓	✓		✓	✓	✓			✓	✓			✓			
Visualizing interaction zones (Rehman et al., 2005)	✓	✓			✓	✓			✓	✓			✓			
"Just-in-time chrome" (Wigdor and Wixon, 2011)		✓			✓	✓			✓	✓			✓			
OctoPocus (Bau and Mackay, 2008)	✓	✓			✓	✓		✓	✓	✓			✓			
ShadowGuides (Freeman et al., 2009)	✓	✓			✓	✓		✓	✓	✓			✓			
TouchGhosts (Vanacken et al., 2008)	✓	✓			✓	✓		✓	✓	✓			✓			
Situations (Dey and Newberger, 2009)		✓	✓	✓	✓	✓	✓	✓		✓			✓	✓	✓	
"What if?" (Lim and Dey, 2009, 2010)	✓			✓	✓	✓	✓	✓		✓			✓			
a CAPpella (Dey et al., 2004)				✓			✓			✓						✓
ICAP (Dey et al., 2006)				✓			✓			✓						✓
Jigsaw editor (Rodden et al., 2004)					✓		✓			✓						✓
Feedforward Torch (Vermeulen et al., 2012)	✓			✓	✓	✓	✓	✓		✓		✓	✓			
PervasiveCrystal (Vermeulen et al., 2010)			✓	✓	✓		✓	✓		✓			✓	✓	✓	
Visible Computer + Slow-Motion Feedback (Vermeulen et al., 2009, 2014)		✓		✓		✓			✓	✓	✓		✓	✓		
Proxemic Flow (Vermeulen et al., 2014)	✓	✓	✓		✓	✓		✓	✓	✓			✓			

Table 3.7: Overview of the different techniques represented in the design space for intelligibility and control.

INITIATIVE Most systems that push intelligibility to the user (*system* initiative), also appear to be embedded and domain-specific. We argue that these techniques tend to be fairly specific to a certain type of application, since they need to have a good idea of when the user requires information, a problem that is challenging to solve for the general case (see Section 3.3.4). In our own work, we explored both system-driven as well as user-driven techniques (e.g., PervasiveCrystal and the Feedforward Torch). We believe a combination of both techniques can be valuable. However, through our experiments with the system-driven Visible Computer prototype, we learned a lot about the difficult balance of providing useful information while avoiding to overwhelm or confuse users (see Section 6.3.5). We tried to apply these lessons in the design of the system-driven Proxemic Flow prototype (more on this in Chapter 8).

MODALITY It is apparent that most techniques rely on the visual modality. There are only a few systems that provide intelligibility through other means, and even these are usually providing visual information as well. Also in our own work, we mostly focused on visual information, although we did explore other modalities with the Visible Computer and the Feedforward Torch. The Feedforward Torch, for example, vibrates to draw the user's attention. Although we believe that a combination of several modalities can be useful, based on our experiences, we still see the visual channel as the primary channel designers should target to support intelligibility.

LEVEL OF CONTROL With respect to control, almost all techniques are on one of the opposite ends of the spectrum. Most systems only provide intelligibility (e.g., Google Maps, OctoPocus) without control mechanism. On the other end of the spectrum, there are very powerful systems such as Rodden's jigsaw editor (Rodden et al., 2004), IOS Cheverst et al. (2005) and a CAPpella (Dey et al., 2004). In our own work, our major focus was on intelligibility, and less so on control. Nevertheless, with PervasiveCrystal, we covered a fairly broad control spectrum, and observed that redundant strategies to control the system were appreciated by users. We believe one of the biggest remaining challenges is in allowing end-users themselves to program their environments, one that will only become more important with the rise of context-aware devices and services in our domestic environments (Mennicken et al., 2014). Existing techniques that support *programmability*, usually employ an external interface and are very general. One could argue whether these techniques are really usable by non-technical users.

3.5 CONCLUSION

In this chapter, we provided an overview of several existing techniques to support intelligibility and control, and introduced a design space based on six dimensions. As a tool for designers, this design space serves two main purposes: (1) it can be used to analyse, compare and relate different existing and future techniques, and (2) given a specific problem, it can be used to generate and iterate over different design

alternatives for supporting intelligibility and control. Designers could, for example, take a system-driven, embedded technique that requires visual attention and think about alternative techniques for that purpose—for example, one that is user-driven with an external interface and relies on haptic feedback.

This dissertation provides an in-depth exploration of the timing dimension, given the lack of general techniques to provide intelligibility and control before, during and after actions. We propose three general techniques along this dimension: *feedforward* before actions (Chapter 5, illustrated with the Feedforward Torch in Section 5.6), *slow-motion feedback* during actions (Chapter 6, illustrated with the Visible Computer in Section 6.3), and *why questions* after actions (Chapter 7, illustrated with PervasiveCrystal). Finally, we describe the design and implementation of ProxemicFlow (Chapter 8) as a case study in supporting intelligibility and control for proxemic interactions, a subdomain of context-aware computing that exhibits many of the challenges described earlier in Chapter 2 (see e.g., Greenberg et al., 2014). In this case study, we cover the full timing dimension.

Before we delve deeper into the three techniques and case study (Chapters 5–8), we describe an exploratory study of a proactive, context-aware mobile guidance system aimed at aiding nurses in their daily care-giving routines (Chapter 4). This study illustrates the impact of design choices covered by the design space and provides additional insights into the issues people face when interacting with context-aware systems.

4

EXPLORATORY STUDY OF A CONTEXT-AWARE GUIDANCE SYSTEM FOR NURSES

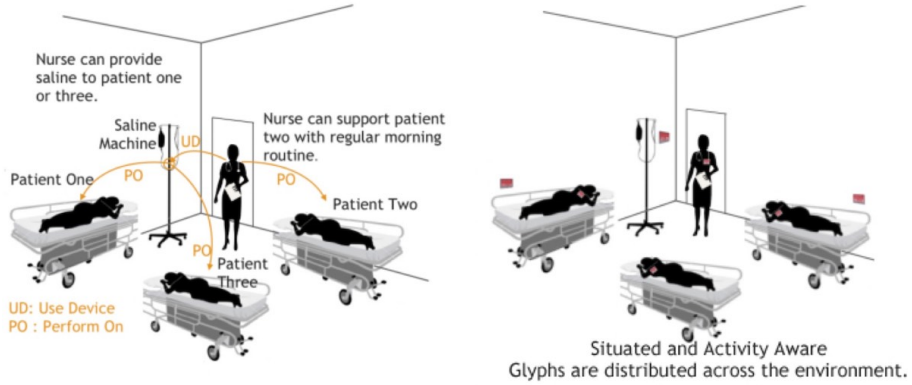
4.1 INTRODUCTION

In this chapter, we describe an exploratory study of a proactive, context-aware mobile guidance system aimed at aiding nurses in their daily care-giving routines. Our goal with this study was to get insights into the issues people face when interacting with context-aware systems, and to get a better understanding of how different design choices—such as the level of control—have an influence on these interaction challenges (see also Chapter 3). In particular, we decided to study a context-aware system in a demanding work environment, as the specifics of the user's job place serious demands on the acceptability of context-aware technologies. Certain design decisions such as the use of proactive or autonomous behaviour might not be acceptable in these environments. Context-aware systems and ubicomp technologies in general are increasingly being deployed in hospital environments to support clinicians in their daily work (e.g., Bardram et al., 2006; Favela et al., 2007). Even though *pervasive healthcare* is gaining more acceptance, these systems sometimes suffer from problems regarding uncertainty in context recognition. For example, during a three month deployment of context-aware technologies in a hospital, Bardram et al. (2006) found that clinicians rejected all *active* context-aware features, suggesting that it is very important for clinicians to *stay in control*.

Previous studies have found that medical personnel would benefit most from having specific information (e.g., guidelines) available about their current activity, linked to medical equipment, places and patients relevant to this activity. However, information in hospitals tends to be still tied to traditional manual (or digitized) patient record files that are often not accessible when caregivers are attending to patients. To address these problems, we developed *situated glyphs*, context-aware micro-displays that show real-time, task-specific information and can be deployed in healthcare environments (Kawsar et al., 2011; Vermeulen et al., 2012a). In this chapter, we first describe the situated glyphs system. Next, we explain our study setup: relevant information categories were first collected in a formative study at the hospital ward (Altakouri et al., 2010), after which we performed a simulation of the situated glyphs system with six nurses. Nurses were introduced to two different hardware prototypes, with different levels of control and different ways of presenting information. We conclude with the findings of semi-structured interviews with the nurses to identify what level of control would be appropriate, and what hardware prototype was preferred.

4.2 SITUATED GLYPHS: PROVIDING ACTIVITY-AWARE VISUAL INSTRUCTIONS

Studies have indicated that there is a clear need to present task- and activity-centric information in demanding work places, such as hospitals (Bardram, 2009) or industrial plants (Heyer, 2010). Consider the situation depicted in Figure 4.1a, in which a nurse is presented with multiple care options involving multiple patients and equipment. She might decide to use saline solution with Patient One or Patient Three, or she might decide to support only Patient Two instead. In each case, she requires information that matches her *current activity* and is linked to the *equipment* and *patients* that are relevant to this activity.



(a) (b)

Figure 4.1: A hypothetical nursing care scenario with and without situated glyphs. Situated glyphs present task- and activity-centric information to the nurse.

The system we study in this chapter presents activity-centric information on micro-displays in the form of *glyphs* (Kawsar et al., 2011): simple visual representations of physical activities that represent relationships between people and objects relevant to that activity.

In the field of information visualization, a *glyph* is a single, graphical unit designed to convey multiple data (Ware, 2012). Different parts of the representation or different visual variables (e.g., shape, size, colour) (Bertin, 1983) are utilized to encode different values. An early example of glyphs was shown by Chernoff (1973), who represented multidimensional data through different attributes of human faces (e.g., a nose, eyes). In the literature, glyphs have been used to represent different attributes of documents (Mann, 1999) or for visualising software management data (Chuah and Eick, 1998). Due to their intrinsic capability of representing multiple variables with a single graphical representation, we identified opportunities to explore the use of glyphs for subtly exposing salient information in dynamic work places.

To this end, we believe that glyphs provide an interesting design alternative to present real-time, in-situ information to support multiple interleaved activities in-

volving multiple individuals and different types of equipment in complex workplaces. Accordingly, we have devised *situated glyphs* as graphical units that are situated in time and space—they are visual representations of activities, and are adaptive, mobile and replaceable. Figure 4.1b shows the same situation as explained above, but here the environment is augmented with multiple situated glyphs. In this case, when a nurse approaches a particular piece of equipment or a patient to perform an activity, corresponding glyphs show the information that is relevant to that activity. One of the key functions of situated glyphs is to help people *discover* the activities that can be performed in a given space, at a given time with the devices and objects at hand.

Figure 4.2 shows an example design of a situated glyph to illustrate the concepts. Consequently, the glyph shown in Figure 4.2b corresponds to a 'red' coded nurse's activity of measuring blood pressure with a 'red' coded patient numbered '3', using a 'red' coded blood monitoring device numbered '19' which is available in the south-east direction and working fine. This glyph design is adaptive and dynamically changes its content depending on the activity at hand and the context of the activity. Glyphs are initially abstract, but on approaching an individual or an object, more detail is revealed as shown in Figure 4.2a.

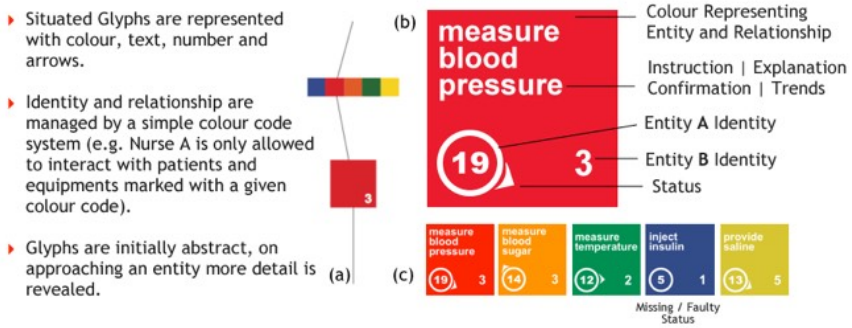


Figure 4.2: An illustrative design of a situated glyph.

Moreover, the spatial distribution of glyphs—the distribution granularity and placement alternatives of situated glyphs—can vary. Situated glyphs can be positioned at different locations in the environment: they can be wearable, portable, or fixed in a certain location. One interesting point of discussion is defining the optimum number of glyphs distributed across the environment. This placement granularity reveals the design trade-off between information capacity and fragmented attention. By increasing the number of glyphs it is possible to present more fine-grained information (Pousman and Stasko, 2006). Additionally, information can then be dispersed across these glyphs in a more situated fashion, i.e., a glyph embedded in an object shows only information about that object instead of showing information about the activity as a whole. However, the caveat of increasing the number of glyphs is that it introduces fragmentation of attention due to the de-

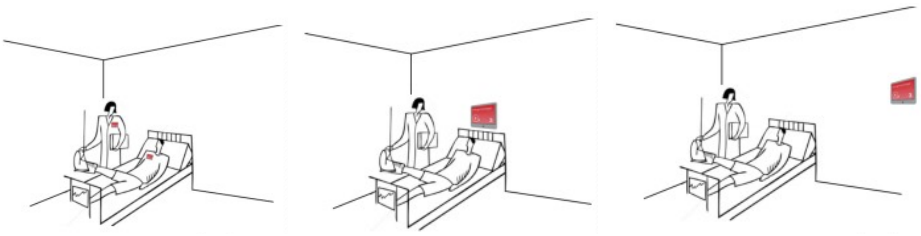
manding context switches which consequently increase the cognitive load of the individuals involved in the activity.

Taking these distribution choices into account, we envision multiple possibilities for the placement of the glyphs. Delving into the 'Situative Space Model' introduced by Pederson (2003), we can logically distribute the glyphs into *manipulable space* and *observable space*. A third alternative is the physical embodiment of a glyph onto an entity. Accordingly, we identify three design alternatives for placement of glyphs:

1. *Entity-centric:* A glyph is embodied in every entity as shown in Figure 4.3a. For individuals these glyphs come in a wearable form, whereas for physical objects, glyphs are embedded in them.

2. *Activity-centric:* A glyph is placed at the location of the activity climax or in the manipulable space as shown in Figure 4.3b. As an example, for an activity involving a patient and a blood pressure monitor, the glyph can be placed on the patient's bed, assuming this activity will be conducted while the patient is in bed.

3. *Space-centric:* A glyph is placed in the observable space and is shared across multiple activities and entities as shown in Figure 4.3c. An example of this kind of glanceable space is the wall between two patients' beds.



(a) Entity-centric glyphs (b) Activity-centric glyphs (c) Space-centric glyphs

Figure 4.3: Different placement possibilities for situated glyphs.

Entity-centric glyphs represent the extreme end of the spatial spectrum. Even though they provide the finest detail of information, they introduce maximum fragmentation of attention in comparison to activity-centric and space-centric glyphs. In addition, compared to the other two alternatives, entity-centric glyphs require fewer adaptations and information updates due to their situated nature.

Technically, such situated and activity-aware glyphs can be realised as a distributed display network. Our first prototype was created using an iPod touch device in an enclosed plastic case, showing only part of its display (Figure 4.4a). The iPod was configured to open the Mobile Safari web browser at a local webpage showing a glyph and could be controlled externally (see also Section 4.3.3). In its latest iteration (Kortuem et al., 2011), situated glyphs form a micro-display network consisting of Jennic JN5139 micro-controllers with µOLED-160-G1 160x128 pixels

(a) (b)

Figure 4.4: Different versions of the situated glyph prototypes. The first prototype (a) enclosed an iPod touch to only show part of the screen. The latest prototype (b) runs on custom hardware and has a physical size of 51 mm × 30 mm.

65k colours OLED displays, as shown in Figure 4.4b. The micro-displays run the Contiki Operating System providing a TCP/IP suite on top of the ZigBee wireless standard.

We studied the usefulness of situated glyphs in a simulation with six nurses of the Geriatric Psychiatry ward at the District Hospital Mainkofen in Germany. In the next section, we discuss the objectives and research questions explored in this study. We also briefly discuss a formative study at the hospital that revealed the most relevant information categories for the nurses. Next, we present the system and tasks that were used during the study to examine the design space. We then delve into the study methodology and explain our procedure for answering the two research questions, followed by both the quantitative and qualitative results of the study.

4.3 USER STUDY

4.3.1 *Objectives and Motivation*

The objective of this study was to investigate how best to present activity-centric information in a hospital environment. We wanted to investigate which methods to provide information are best suited in this kind of environment and under what circumstances. More specifically, this study aimed to answer the following research questions:

Q1 **Degree of co-location**: Should information be presented in an *embedded* or *external* way? In what situation is there a preference for either, and why?

Q2 **Initiative**: Should the initiative lie with the *user* or the *system*? Or is a more hybrid approach more suitable? In what situation is there a preference for either, and why?

Note that each of these research questions corresponds to a dimension in the design space for intelligibility and control that we introduced in Chapter 3. This

study provided us with additional insights into the advantages and disadvantages of these different methods to provide information to users. To answer these research questions, we conducted a study in which nurses were presented with different prototypes along two axes (degree of co-location and initiative). We collected both quantitative and qualitative data, the results of which are discussed in Sections 4.4 and 4.5.

4.3.2 *Results from Formative Study at District Hospital Mainkofen*

Before developing the prototype system, our colleagues conducted a formative user study at a hospital (Altakouri et al., 2010). Both the formative study and the second study took place in the Department of Geriatric Psychiatry of the District Hospital Mainkofen, a specialist clinic near Deggendorf, Germany. Nurses in this department support elderly patients suffering from dementia. In the formative study, a Contextual Inquiry was conducted (Beyer, 1997) to observe and analyse the daily routines of the nurses. The study was primarily focused on identifying the information needs of nurses, rather than on investigating issues when deploying prototypes for this environment. Based on analyzing the results from the formative study, we came to four different categories of information that are relevant for nurses during their daily routines (Kawsar et al., 2011), which we used as a foundation for developing our prototype system:

1. *Identity and Relationship*: This category of information describes the identity of a patient, medical equipment, etc. and their relationship with each other in the context of an activity. This type of information helps nurses to make informed decisions regarding what equipment to use with which patient.
2. *Instructions*: This information category describes guidelines to perform a medical routine with or without specific medical equipment.
3. *Confirmation*: Feedback about successful completion of a medical routine with or without specific medical equipment.
4. *Explanations*: This category of information provides explanations to address exceptional situations e.g., when devices are malfunctioning.

4.3.3 *System Description*

4.3.3.1 *Situated Glyphs Used During the Study*

Figure 4.5 shows the six different types of glyphs we used in the study, which contain information sorted into the four different information categories gleaned from the formative study.

When nurses approach a patient, they are first shown an overview of the tasks they must perform with this patient (Figure 4.5a). The task overview consists of a short task list together with a patient ID (the number 2).

(a) Task overview (b) Single task (c) Confirmation

(d) End confirmation (e) Exception (f) Explanation

Figure 4.5: The different types of glyphs used in the study.

The system or user then proceeds to the first task (depending on the initiative), and is shown a glyph representing an individual task (Figure 4.5b). This type of glyph consists of a short description of a task (in this case: "Blutdruck messen", German for "Measure blood pressure"), the patient ID (the number 2), and the ID of the medical instrument to use for this task (the number 1: the ID of the stethoscope). Patient and equipment IDs correspond to the *Identity and Relationship* information category.

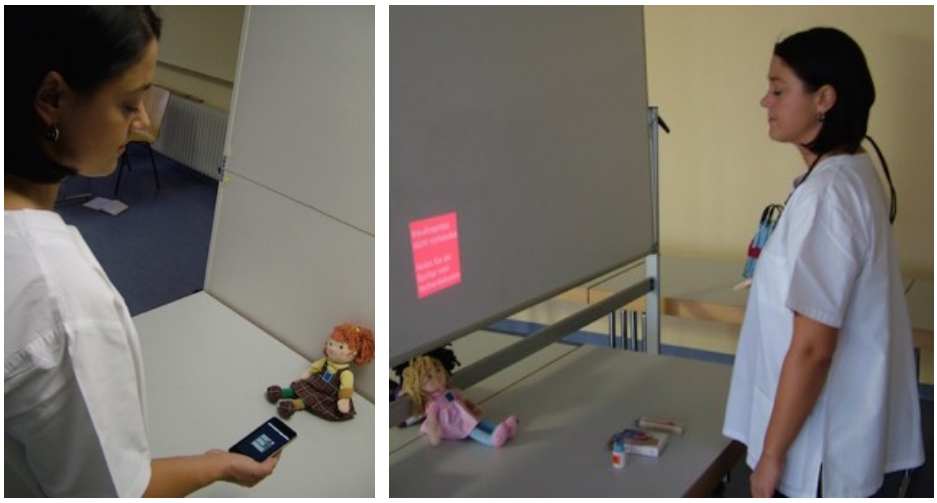
When participants complete a task, the system confirms this by showing a check mark (Figure 4.5c). Finally, when all tasks for a patient have been successfully completed, a last confirmation is shown, consisting of a check mark and the patient ID (Figure 4.5d). These glyphs correspond to the *Confirmations* information category.

When there is an exceptional situation (e.g., a patient is allergic to a certain type of medication), the task glyph will indicate this. For the purpose of this study, we used a single type of exception that reports problems with the medical instrument needed to complete the task. In this case, the circle around the medical equipment ID is coloured red (Figure 4.5e). The next glyph then shows an explanation of the exception to help the nurses solve the problem. In this case, the explanation (Figure 4.5f) tells participants to get the insulin injection from a first aid kit in another part of the room.

4.3.3.2 *Hardware Prototypes*

To investigate the *Degree of co-location* dimension (research question Q1), we used two different hardware prototypes to show glyphs to participants, as shown in Figure 4.6. For the *External* condition, we used an Apple iPod touch handheld (Figure 4.6a) where glyphs are shown on the display. For the *Embedded* condition, we

developed a wearable projector prototype using a plastic case with a mirror oriented at 45 degrees at the bottom and a neck strap at the top which projects glyphs in front of the user (Figure 4.6b), allowing them to focus on their current task.



(a) External prototype (b) Embedded prototype

Figure 4.6: The two prototypes in use. Participants held the external prototype (an Apple iPod touch) in their hand while performing tasks, or they wore the embedded prototype around their neck which allowed them to keep both hands free.

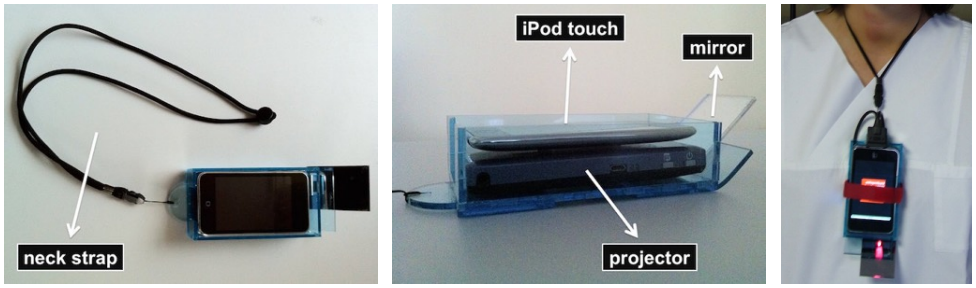
The plastic case contains another Apple iPod touch which is connected to a MicroVision ShowWX™ Laser Pico Projector (Figure 4.7). The embedded prototype weighed about 300 grams in total.

4.3.3.3 *Software Prototype: A Distributed Display System*

We built a distributed display system in order to be able to remotely control the display of glyphs on both the embedded and external prototypes. Both prototypes were configured to open the Mobile Safari web browser at a local webpage showing a glyph. The web page was being served from a laptop running Apache with PHP over an Apple Airport Express Wi-Fi base station inside the room. A specific web page showed the current glyph (as seen by the nurse), while another web page allowed the experimenters to control which glyph was currently being shown on the prototype. Glyphs that were shown on the embedded prototype were automatically mirrored and rotated to appear in the correct orientation when projected.

To simulate context-awareness, we used the Wizard of Oz technique[1] (Kelley, 1984). One of the experimenters (Figure 4.8a) remotely controlled the glyph display

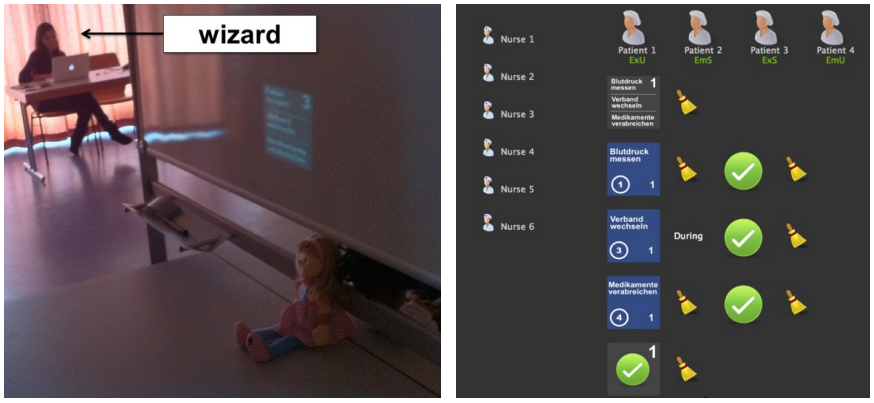
1 With the Wizard of Oz technique, an experimenter (the *wizard*) partially operates an intelligent system, letting the user believe that the system works autonomously. It is commonly used to allow for rapid evaluation without having to overcome all technical difficulties first, since participants can interact with the system as if it was fully functional.



(a) Top view (b) Side view (c) Worn on the body

Figure 4.7: The embedded prototype consisted of a wearable plastic case with a strap, allowing participants to wear the device around their neck. The case contained an Apple iPod touch running our software connected to a MicroVision ShowWX™ Laser Pico Projector and a mirror oriented at 45 degrees to allow the projected image to be displayed in front of the participants.

on the prototypes through the specific controller web page (Figure 4.8b). She started by selecting one of the trials for the experiment, and then showed specific glyphs or cleared the screen. To update the glyph being displayed in the web page on the prototype, we used AJAX calls and HTTP streaming[2].



(a) Wizard (b) Wizard controller UI

Figure 4.8: The wizard controlled the prototypes through a dedicated controller web page.

4.3.3.4 *Initiative: Voice Control versus Automatic Context Recognition*

The dimension *Initiative* (research question Q2) was examined by varying the way participants proceeded from task to task (user-driven versus system-driven interac-

2 HTTP streaming is a mechanism to stream server data in the response of a long-lived HTTP connection.

tion). In the case of user-driven interaction, participants were required to issue the voice command, "Next", after which the next glyph would be shown.

Both system- and user-driven interaction were controlled by the wizard. In the case of system-driven interaction, the wizard would enact automatic recognition of completed tasks by proceeding once she observed that the participant had completed a task. With user-driven interaction, the wizard would proceed to the next glyph when the participant issued the voice command "Next". Although participants could see the wizard, we explained that she was there to make sure the system would run smoothly. None of the participants realized that the wizard was actually driving the interaction.

4.3.4 *Study Methodology*

4.3.4.1 *Room Setup*

We conducted our study in the Department of Geriatric Psychiatry of the District Hospital Mainkofen, Germany. For the course of the study, we rented a room in this hospital where we set up a number of tables and whiteboards to simulate two different patient rooms, with two patients each. Participants took part in the experiment in between their shifts, as we wanted nurses to be in the flow of their daily work activities. Instead of requiring patients to be present in the room during the study, we used dolls to serve as stand-ins for patients and toy equipment instead of real medical equipment, as seen in Figure 4.9. Participants were asked to role-play their usual behaviours as professional nurses on these dolls using the toy equipment.

There are a number of reasons why we relied on dolls instead of real human patients. First of all, there are several ethical issues involved in using human patients during the study, such as privacy concerns and the inability to guarantee their well-being—as several patients at Mainkofen suffer from mild to severe cases of dementia and might get confused or anxious during the trials. Second, our study was mainly focused on supporting nurses in their daily activities, and not on the patients. Finally, as part of their training, nurses are accustomed to practice medical routines on training dolls (Lapkin et al., 2010).

We used tables and whiteboards to simulate two different rooms. Figure 4.10 shows the layout of the left room, which consisted of two tables with a doll on each side, representing two patients in hospital beds (Figure 4.10a). During the initial formative study in Mainkofen, nurses reported that they usually wheel a cart with medical equipment from room to room. To simulate this situation, we collected all toy instruments on a chair in the middle of the room before each trial (Figure 4.10b), providing nurses with easy access to the necessary equipment. The right room (which is partly visible on the right side of Figure 4.10) was set up in exactly the same way. The left and right room were separated by the whiteboard in the middle (representing a wall).

Figure 4.9: The apparatus used for the study: a variety of toy medical instruments (e.g., a stethoscope and manometer, an injection needle, bandages) together with dolls that served as stand-ins for patients. All objects were numbered and tagged with RFID tags (coloured square stickers).

4.3.4.2 *Participants and Design*

Initially, we recruited 10 nurses from the Geriatric Psychiatry ward at Mainkofen, which consists of 17 nurses in total (4 male, 13 female). However, 4 nurses dropped out of the study due to an incompatible shift work schedule, leaving us with 6 nurses in total. These 6 nurses (1 male, 5 female) participated in the study over the course of approximately a week, and performed the experiment in-between their shifts. Participants ranged from 27 to 46 years old, with a mean age of 34. In a demographic survey that was conducted at the end of each session, all participants reported owning a mobile phone. They rated their experience with computers and mobile phones as average, ranging from 2 to 4 on a five-point Likert scale, with a mean expertise of 3 out of 5 (5 = expert, 1 = none). Five out of six participants owned simple phones (not smart phones), while one nurse used an iPhone. A complete study session lasted for approximately 2 hours and nurses were paid 50 EUR for their participation.

The study was set up as a 2×2 repeated measures within-subject design (Degree of co-location x Initiative). The independent variables were the *Degree of co-location* with two levels (External and Embedded) and *Initiative* with also two levels (System and User). The factorial design produced 4 different trials per participant, one for each combination of the *Degree of co-location* and *Initiative*.

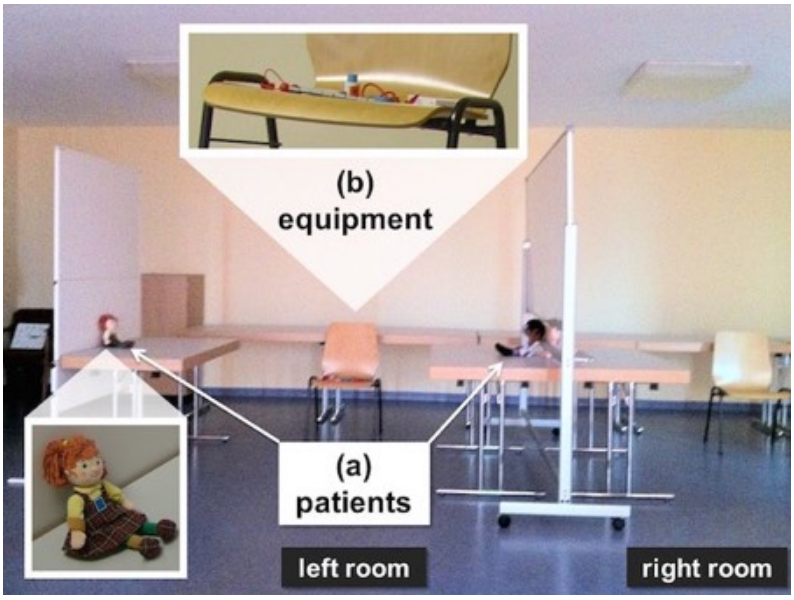


Figure 4.10: The setup of the rooms: two beds with patients (a table with a doll) on opposite sides of the room, and a cart with medical equipment (a chair with toy instruments) in the middle of the room.

During each trial, subjects performed 3 medical procedures on one of the 4 patients (represented by 4 dolls). Participants always moved between patients in the same order, but the 4 different conditions were assigned to patients in random order to minimize learning effects. Figure 4.11 gives an overview of the medical procedures subjects had to perform during the study. These 6 tasks were randomized over the 4 patients, but were kept stable between participants so that each participant had to perform the same tasks with a certain patient (albeit possibly with a different condition).

4.3.4.3 *Procedure*

Before each session, the researchers prepared the handheld and wearable projector prototypes that would be used for investigating each of the four conditions. After obtaining informed consent and explaining the basic setup and goal of the study, participants were guided through a trial session with the *External–System* condition, to give them an idea of how the system worked. Participants were told that exceptions might occur during the trials, which would be indicated with a red circle around the medical equipment ID (see Figure 4.5e). We also told participants that the system would in this case instruct them what to do in order to solve the problem.

Participants would then start with the first trial. As mentioned before, subjects performed 3 medical procedures on one of the patients during each trial. A trial

A1. Measuring blood pressure
A2. Changing a bandage
A3. Measuring temperature
A4. Measuring sugar level
A5. Administering medicine
A6. Giving an insulin injection **[exception]**

Patient	Pa1	Pa2	Pa3	Pa4
	A1	A1	A3	A1
	A2	A4	A2	A4
	A5	A6	A5	A3

Figure 4.11: The different medical procedures participants had to perform during the study and their allocation to the four different patients.

ended when all tasks had been successfully completed. After each trial, we asked participants to answer a questionnaire and conducted semi-structured interviews to obtain both quantitative and qualitative feedback. The questionnaire included general questions about the usability of the prototype, based on the IBM Computer Usability Satisfaction Questionnaire (Lewis, 1995) and the NASA Task Load Index (Hart and Staveland, 1988). When a participant had completed all four trials, we gathered demographic data and conducted a final semi-structured interview in which we asked questions primarily focused on comparing each of the four conditions. The questionnaires used for the study can be found in Appendix B.1.

The entire study was video recorded, we took photographs of participants working with our prototypes and used a voice recorder to document all interviews. The voice recordings were transcribed and photos and images inserted into the transcript. The results were collated and analysed by the researchers to understand the data.

4.4 QUANTITATIVE RESULTS

We compared the scores for each of the different conditions over both the IBM Computer Usability Satisfaction Questionnaire (IBM CUSQ) and NASA Task Load Index (NASA TLX) questions in the post-trial interviews. Mean results for each of the post-trial interview questions can be found in Figure 4.12 and Figure 4.13. Note that we were only able to perform the study with 6 participants since several nurses dropped out of the study. Consequently, we do not claim that these results are statistically significant or free from confounding factors caused by the small sample size. Nevertheless, we still include them here, as they indicate general trends that we discuss further in the qualitative analysis (Section 4.5).

The results indicate that *External–System* consistently scores highest (mean = 1.17, $\sigma = 0.21$) on a five-point Likert scale (IBM CUSQ: 1 = strongly agree, 5 =

strongly disagree; NASA TLX: 1 = very low, 5 = very high); followed by *Embedded–System* (mean = 1.77, $\sigma = 2.00$); *External–User* (mean = 1.95, $\sigma = 0.59$); and finally *Embedded–User* (mean = 2.14, $\sigma = 0.57$). Participants found the embedded prototype uncomfortable (Figure 4.12) and physically demanding (Figure 4.13), resulting in low scores for these respective questions. This was mainly due to the specific form factor of the embedded prototype and the fact that projection requires fairly low light conditions. The user-driven prototypes were also generally ranked worse by participants, and this mostly because it was cumbersome for participants to have to drive the interaction using voice input.

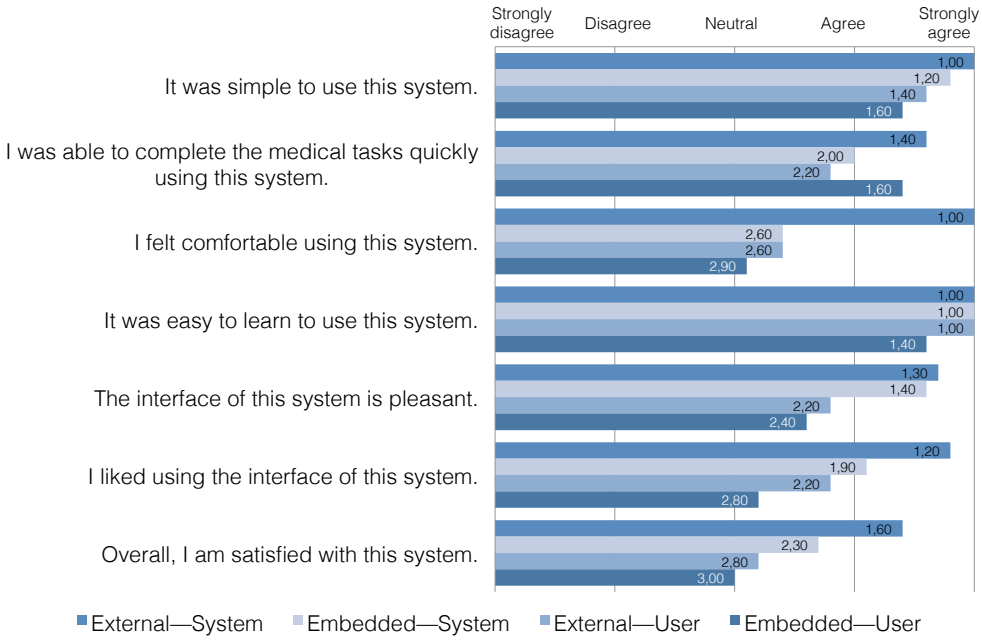


Figure 4.12: Results based on questions from the IBM Computer Usability Satisfaction Questionnaire for all four conditions.

At the end of the study, participants were asked to choose between *Embedded* versus *External* and *User* versus *System* respectively. Four out of six participants preferred *External* above *Embedded*. The same four also preferred *System* over *User*, indicating a divide between *External–System* (4 participants) and *Embedded–User* (2 participants), with a general preference for *External–System*. To better understand participants' motivations for these scores and answer our two research questions, we now discuss the qualitative results of the study.

4.5 QUALITATIVE RESULTS

The semi-structured interviews with the participants were transcribed and coded, after which they were analysed and discussed by two of the researchers. The anal-

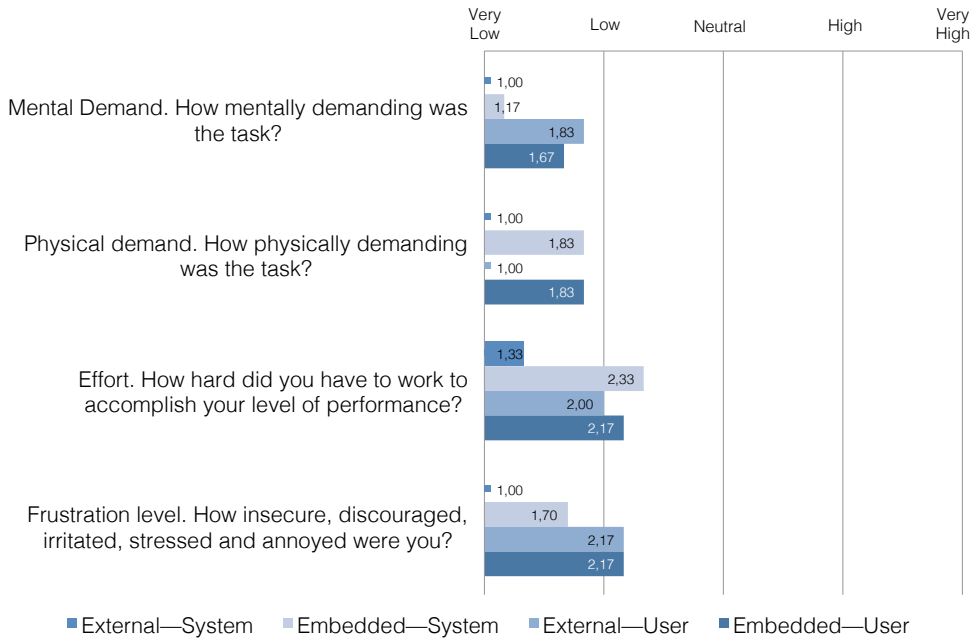


Figure 4.13: Results based on questions from the NASA Task Load Index for all four conditions.

ysis of these interviews combined with the quantitative findings revealed several insights regarding nurses' preferences for the alternative designs, qualitative assessments of the prototypes and the overall user experience. In what follows, we go over the qualitative results in the order of the two research questions.

4.5.1 *Use Attachable Displays to Present Real-time, Activity-centric Information*

The first research question (Q1) we sought to answer examined whether patients would prefer to use a handheld device for receiving information while performing medical routines, or rather rely on a wearable, personal projector. We anticipated both advantages and disadvantages for these different form factors. While a mobile device is easier to carry around, it does require focused attention and might therefore force the nurses to occupy themselves with the device instead of the patient. Although a wearable projector would allow users to keep both hands free for interacting with the patient while displaying information in-situ where it is needed, it could be annoying to wear and requires specific lighting conditions. We were interested in discovering how nurses weighed the disadvantages of each form factor against its advantages and which one they preferred under which circumstances.

Participants in general felt that the wearable projector in the *Embedded* conditions constrained them in their work activities, as they had to be aware of the projector at all times. This was found to be mainly due to the device being relatively heavy, and

not being fixed to the participant's body. When nurses had to lean towards a patient, the device would dangle around their neck and sometimes twist and turn towards their body, causing the projected image to be shown elsewhere, as shown in Figure 4.14. Another major issue that was reported during the interviews was the need for low-light conditions, which might be unsuitable in patient rooms, especially during daytime or when performing more delicate medical routines such as injecting insulin. Additionally, nurses mentioned that there are several situations when there would be no suitable projection surface available. They were also worried about the projector blinding patients when facing them. Finally, several participants reported they were afraid of their own safety and those of the patients when wearing a device around their neck in psychiatric departments. Patients in these departments might be aggressive at times and try to grab and pull the device, which could strangle the nurse. For this reason, nurses also refrain from wearing necklaces during duty. Moreover, they were also worried about the patients' safety while performing routines (e.g., scratching patients with the device).

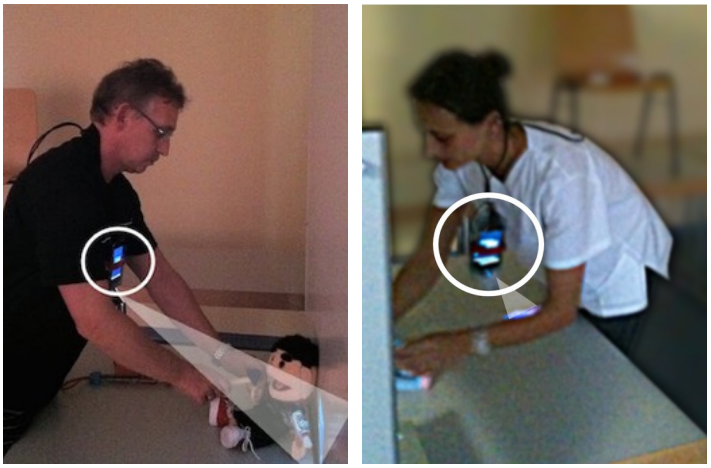


Figure 4.14: Nurses felt they had to be aware of the wearable projector at all times, as it would dangle and sometimes twist and turn when they were leaning forward, causing the projected image to be displayed elsewhere.

These comments were also reflected in the quantitative results, where participants generally rate the physical demand for the embedded prototypes higher than for the external ones (mean = 1.83 versus 1.00; median = 2.00 versus 1.00; $\sigma = 0.937$). To verify whether the reported issues were due to the specific limitations of our employed prototype, we also asked participants whether they could imagine using either a similar system that employed steerable, ceiling-mounted projectors, or a wearable projector the size of a stamp that would be attached to their uniform. Most participants again rejected both suggestions, due to the need for low-light conditions and a suitable projection surface. Moreover, they were afraid that a stamp-sized projector might still hurt patients while performing care-giving activities.

While we anticipated that nurses would experience some difficulties with the wearable prototype, we expected them to appreciate the fact that they would have both hands free for performing routines. Even though most participants confirmed that this is indeed very important, four out of six nurses still preferred the handheld prototype. While they did mention that it was also important to be able to move freely—which was more difficult with the wearable projector—we were still surprised that the inability to use both hands did not have a stronger impact on their preference for the handheld prototype. During the interviews and analysis of the photographs and video recordings, however, we noticed that all participants actually managed to keep both hands free during most of the trials with the handheld prototype. Participants intuitively placed the handheld on the table, waited for instructions in between tasks, and would occasionally glance at the display while performing routines, as seen in Figure 4.15. Two participants (P1 and P5) did prefer the wearable prototype over the handheld one, even though they also tended to put the handheld prototype on the desk in front of them. These participants were mainly concerned about where to put the handheld device, which was also confirmed by other nurses. P1 stated: "I do not want to have to think where I should put the handheld". P5 also agreed: "Where should I put the iPhone? I can't just place it somewhere, patients might take it away." P5 especially liked being able to use the projector to share information with the patient (e.g., what she would be doing, the patient's progress).

Figure 4.15: Participants often placed the mobile device in front of them in order to still have both hands free to do the activities.

Given the tendency of nurses to keep both hands free, we also asked participants' opinion on having a fixed display next to the patients bed. Nurses were mostly concerned about permanently fixing displays to the wall in their ward because aggressive or disoriented patients might break or misconfigure the device. Moreover,

P3 mentioned that beds are moved all the time, which would render some of the displays useless. When nurses were asked about a hybrid system that combines a handheld device and a fixed display, by attaching handhelds to designated spots on the wall when administering patients, three out of six nurses reacted very positively. P6 said: "It would be the ideal system.", while P5 mentioned "This would be super!". The three other nurses were less confident about their opinion. They also reported that a hybrid system could certainly be useful, but mentioned they would need to try it out first.

Based on the results of our study, we feel that although personal projection certainly has its uses (Rukzio et al., 2012), the technology would probably be too invasive for use in hospital environments. Nurses were especially concerned about the need for low-light conditions and for a suitable projection surface. They also worry about wearing devices on their body, both for safety reasons and because they want to be able to move freely. Regarding the handheld prototype, nurses did not like the fact that they had to find a place to put the handheld while performing routines, and were concerned about patients damaging or taking away the handheld. They generally preferred to put unnecessary devices or instruments away when they were dealing with patients to guarantee their own safety and that of the patients. Given these observations, we therefore recommend against personal projectors (especially wearable ones), and propose a hybrid solution where nurses carry mobile devices that they can attach to designated spots on the wall while administering patients. All nurses saw value in this approach, and three out of six nurses were very positive about the idea. Further studies will be necessary to confirm the value of this solution.

4.5.2 *Allow Nurses to Switch to User-driven Interaction*

The second research question (Q2) dealt with which type of initiative (system-driven versus user-driven interaction) users would prefer under which circumstances. To our surprise, the majority of participants (four out of six) preferred system-driven interaction (*System*) over user-driven interaction (*User*). Several nurses reported that they found system-driven interaction easier to use, as they did not have to think too much about what to do next. During the semi-structured interviews, we asked participants whether there would be situations in which they would need to override the system. P3 argued it would be necessary to be able to skip certain medical procedures that they were assigned to perform to a patient, either to perform the tasks at a later time or skip them completely when they would have been rendered unnecessary. Even the participants who preferred system-driven interaction, reported that there should always be a means for overriding the system. Moreover, several nurses stressed the importance of being able to cope with unexpected situations and emergencies, as interruptions often occur. P6 mentioned she needs to call a doctor when she notices something unusual (e.g., out of the ordinary vital signs), and might be called away herself at any time to assist doctors or other nurses (e.g., when a patient gets aggressive).

4.5.2.1 *Allow Users to Determine the Pace*

The nurses that preferred user-driven interaction (P1 and P5), both mentioned that they did so because they wanted to *control the speed of moving between tasks themselves*. System-driven interaction would sometimes be too slow or too fast for them. Two other nurses—who generally preferred system-driven interaction—also stressed the importance of being able to control the pace. The most experienced participant (P1, aged 47) had a very strong preference for user-driven interaction, as he argued that he knew exactly what to do and wanted to proceed to the next task as quickly as possible. P1 was also the fastest in completing all four trials. The pace at which nurses move through routines they have to perform can vary depending on factors such as their experience, age, or the specific patient and situation (e.g., during emergencies nurses have to act quickly, and cannot afford to wait for the system). During one of the user-driven trials, P1 argued: "I like the fact that the system is fast, it follows my own pace." For one of the system-driven trials, he said: "It was too slow, I had to wait too long. It is very important to me that the system works at my usual speed." P2 reported waiting for confirmations slowed her down: "I know how to do a task, I don't like having to wait for a confirmation. My pace is higher than that of the system." We recommend to allow users to drive the interaction when they need to in order to provide the flexibility that is needed to cope with the broad range of diverse people and unexpected situations in hospitals.

4.5.2.2 *The Level of Interaction Required Defines Who Drives the Interaction*

As mentioned before, nurses generally liked the idea of having relevant tasks be displayed automatically in a system-driven way, when the system detected that the nurse was in the vicinity of a certain patient. The need to control the pace only came into play when there were multiple tasks to do. In this case, nurses wanted to have the means to mark a task as complete and move to the next one, if necessary. We therefore argue that, when showing activity-centric information, the level of interaction required is a defining factor for deciding between user and system initiative. When there is no interaction required (e.g., a single task for a particular patient), we recommend system-driven interaction, as this will minimize the mental load of the nurses. When nurses have to perform a multi-stage process (such as when there are several routines to be performed on a patient), we recommend giving the nurses the possibility to drive the interaction, as this would allow them to determine their own pace and accommodate unexpected situations (e.g., when a patient's blood pressure suddenly drops).

4.5.2.3 *Preference of Self-Reporting over Activity Recognition*

Since nurses perform several common routines on a daily basis, they are often in a better position to recognize when a task has been completed than the system. P1 mentioned that he just wanted to tell the system to check off an item from the patient's list, and move on to the next task. Trained personnel such as nurses are much better qualified to make accurate decisions about which procedure has

been completed and which one ought to be performed next. We therefore argue that nurses should be able to report to the system which procedures have been performed, instead of relying on automatic activity recognition, as the accuracy is likely too low (Favela et al., 2007) and the cost of the system making mistakes will be too high to warrant a fully autonomous system. We do believe it would be useful for the system to make suggestions, as participants generally reported that they liked the idea of having tasks displayed when they approached patients. This is in line with a study by Bardram (2009), in which he argues that it makes sense to *link* specific context events—such as the physician's location in front of the patient's bed, or the presence of specific objects such as a certain patient's pill tray—to a relevant activity.

4.5.3 *Task Overviews and Completion Confirmations are Key Information*

Participants were extremely positive about being shown an overview of tasks that had to be performed when they approached a certain patient. Nurses have to document their activities on a PC, and usually have to do so after the facts, during a quiet moment. All participants felt it would be great to have a system available that automatically logs the routines that they perform. P5 stated: "Having the system automatically document my actions would be great, it saves time." P2 called the idea "Super! It's nice to see that people are trying to ease the job of nurses. But the system should help us, and not draw our attention away from the patients." P4 said: "It is nice to know what to do. It lightens our work significantly. It is nice to know you won't forget something when you're with a patient." Finally, P2 and P4 both mentioned they would have more time for their patients with this kind of system.

Additionally, participants especially appreciated the confirmations at the end (Figure 4.5d), while results were less conclusive about the individual task confirmations (Figure 4.5c). Some participants even found the task confirmations annoying. P2 said: "Confirmations in-between are confusing".

4.6 DISCUSSION

Drawing on the results of this study, we now describe several insights we gained about the requirements for deploying context-aware systems in demanding work environments such as hospitals. We relate these observations to the general theme of this dissertation: addressing interaction challenges in context-aware systems by making these systems intelligible and controllable.

First, with respect to the level of control, we noticed that participants generally preferred the design in which the system was driving the interaction. Even so, participants also indicated that they needed a certain level of control and flexibility, suggesting that it is still necessary for this kind of system to provide end-user control. Especially given the frequency of exceptional situations, it is essential having a means to override the system. A dynamic level of autonomy might be most appropriate, as several participants reported this as being 'ideal'. Systems in healthcare

situations should strike a delicate balance between having control over the system and providing ease of use by having the system automatically perform actions on behalf of the clinicians. We suggest that, in combination with system-driven interaction, nurses should be allowed to intervene and take control at all times.

Even though nurses generally indicated a preference for the handheld prototype, we noticed during the interviews that having both hands free for performing activities is a key requirement for this kind of system. This was also evident because many participants actually placed the handheld device on the table (see Figure 4.15). Moreover, the results also indicate that deploying wearable systems and relying on projection in hospital environments can introduce several problems. Nevertheless, given the hands-free requirement, we argue that an alternative *Embedded* design, where information is displayed in situ, would be most appropriate for these systems that provide real-time, activity-aware instructions. Based on the results gleaned from semi-structured interviews in this study, we argue for attachable, mobile displays that nurses can carry around and attach to a surface when approaching a patient in the next iteration. This way, nurses could reap the benefits of having both hands free, without the limitations of wearable systems or projected imagery. A follow-up study in a different ward, later confirmed that nurses indeed preferred the attachable mobile display design (Vermeulen et al., 2012a) over a projection-based or handheld-only system.

Finally, the study suggested that feedback is crucial. Nurses prefer to gather information as soon as possible, to be well prepared and maximize their efficiency. More specifically, task overviews and confirmations that the system recognized an activity or that all tasks for a specific patient were performed are considered to be essential information. Participants were also very enthusiastic about the benefits of automatically documenting their actions, which is currently a tedious task which they have to perform after the fact.

4.7 CONCLUSION

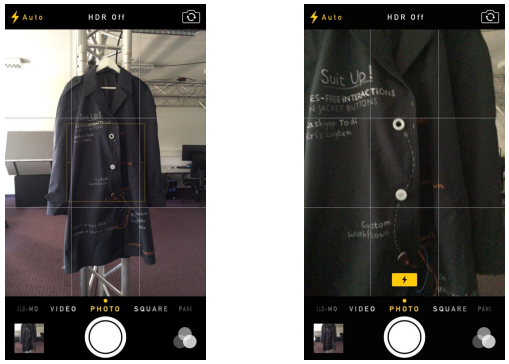
In summary, in this chapter we conducted a study of a context-aware prototype that provides activity guidance for nurses in a geriatric psychiatry ward. Based on a lab study with nurses and detailed semi-structured interviews, we extracted a number of insights regarding interaction challenges of context-aware systems in demanding work environments. We concluded that it is important to strike a balance between ease of use through autonomous behaviour and control by allowing users to take over at all times. With these findings, we provide additional anecdotal evidence for the interaction challenges discussed Chapter 2. The study also provided us with additional understanding with respect to how the alternatives for the *degree of co-location* and *initiative* dimensions in the design space compare (see Chapter 3). Although we concluded that an embedded approach where activity guidance information was presented in-place was most appropriate, we also observed that wearable computing and mobile projection technologies might be difficult to integrate in hospital environments.

5

THE DESIGN PRINCIPLE FEEDFORWARD

5.1 INTRODUCTION

As discussed earlier (see Section 2.3.3), the design principle *feedforward* can be an effective way to bridge Norman's gulf of execution—the gap between the user's intentions and the allowable actions—as it *tells users what the results of their actions will be*. If users know what they can expect, they can make an informed decision about what actions need to be performed to achieve their goals. A very basic example of feedforward is an informative label on a button, which tells users what will happen when they push it. However, feedforward exists in many forms and guises. Consider, for example, how the iPhone uses a small icon to indicate that the camera flash will go off when taking a picture in low light conditions (Figure 5.1). Knowing this can be helpful to avoid triggering the flash in situations where its use would be inappropriate (e.g., in the cinema or at a concert). And indeed, this is feedforward, as it gives users information about what will happen when they press the shutter button. In fact, Wensveen (2005) would classify this as an example of *augmented feedforward*, which we will explore further in Section 5.4.2.



(a) Flash will not go off (b) Flash will go off

Figure 5.1: When the flash is set to auto (top left corner in both figures), the iPhone shows a yellow flash icon in low light conditions (b) to indicate that the camera flash will be used (source: Apple iOS 7).

We argue that feedforward is one of the key design principles for dealing with complexity in interaction 'on the execution side', as it helps users know what to expect and what to do to achieve their goals (Norman, 2013b). Although it is tempting to argue that we should avoid complexity altogether, Norman (2011) stresses that complexity is in some cases unavoidable and should rather be tamed through good

design. The more complex the interaction, the larger the need for well-designed feedforward. Factors that contribute to this complexity include dynamic behaviour (which reduces predictability), a lack of discoverability or complex rules of behaviour (which make it harder to form a correct mental model)—common problems when interacting with ubicomp systems (Rehman et al., 2002). Consequently, feedforward will also have an important role to play in the design of context-aware systems and appliances, and ubicomp environments in general (see Section 2.3.3). Indeed, as mentioned earlier, Bellotti and Edwards (2001, pg. 203) include feedforward among the four design principles that are necessary to make systems intelligible.

Unlike the notions of *feedback* and (*perceived*) *affordances*[1] (Norman, 1988), which, over the past few decades, have emerged as core concepts in interaction design, *feedforward* is not well-known among designers. Even so, almost every designer has—in one way or another—already used feedforward in their designs. Yet, opportunities for feedforward are often left unexplored because designers are not fully aware of this design principle. An important factor contributing to this limited awareness is the lack of a precise and generally accepted definition of feedforward. Furthermore, there is no existing library of examples and proven solutions for applying feedforward that designers can rely on.

In this chapter, we address those problems. First, we provide a new definition: we reframe feedforward by analysing and comparing existing definitions, and further disambiguate it from the related design principles *feedback* and *affordances*. Secondly, we discuss several existing examples of feedforward and clarify what is—and what is *not*—feedforward. Finally, we identify four new classes of feedforward: *hidden*, *false*, *sequential*, and *nested* feedforward. We begin our exploration with a short historical background on feedforward and how it has been used before.

5.2 BACKGROUND

The term *feedforward* originated in control theory (e.g., Meckl and Seering, 1986). In this domain, feedforward is also about predicting what will happen: feedforward control systems generate inputs to attain a certain desired output based on a predictive model of the system. An advantage of feedforward control is increased performance over systems that rely solely on feedback, with applications such as engine-torque control (e.g., Iwasaki and Matusi, 1993) or speeding up disk seeks (e.g., Atsumi, 2009). In addition, the term feedforward has also been used in artificial neural networks (e.g., Hornik et al., 1989), to denote neural networks where information only flows in one direction (i.e., where there is no feedback between the different layers).

1 Norman later argued for replacing the term 'perceived affordance' with 'signifier' to avoid confusion (Norman, 2008, 2013b). However, for historical relevance and to accurately reflect what others have written about the relation between feedforward and perceived affordances, we will continue to use 'perceived affordance' throughout this chapter. All mentions of this term can, however, be replaced with 'signifier'.

In this chapter, however, we focus on the notion of feedforward in the context of interaction design. An early reference to user interfaces that allow the user to predict what will happen, is Nielsen's notion of *prospective feedback* (Nielsen, 1993). When discussing the Eager system (Cypher, 1991)—which automates repetitive tasks by observing the user, i.e., an advanced form of automated form filling—Nielsen notes that Eager "provides the user with information about what it will do before it does it". By showing what interface elements it will operate on next, Eager allows users to first verify that its proactive behaviour is desired. When users are then confident about Eager's inferences and the resulting outcome, they can allow Eager to go ahead and complete the task. Nielsen notes that "such prospective feedback is likely to be a necessary usability principle as long as this type of system does not have perfect inference capabilities". This also reaffirms the importance of feedforward for context-aware systems (Bellotti and Edwards, 2001), which—like Eager—exhibit dynamic behaviour and take actions on the user's behalf (see Section 2.3).

The first definition of feedforward in the context of user interface design was given by Djajadiningrat et al. (2002, pg. 285). They define feedforward by contrasting it with related concepts such as feedback and perceived affordances. Perceived affordances refer to information available in the design that allows the user to perceive action possibilities (Norman, 1999)[2]. Note that Djajadiningrat et al. (2002) state that, unlike feedforward, perceived affordances do not communicate the purpose of an action [emphasis ours]:

> We distinguish between information before the user carries out the action (pre-action), and after the user carries out the action (post-action). These phases correspond with feedforward and feedback. *Feedforward informs the user about what the result of his action will be*. Inviting the appropriate action is a prerequisite for feedforward but it is not sufficient. The product also needs to communicate what the user can expect. Feedback informs the user about the action that is carried out, shows that the product is responding, indicates progress, confirms navigation, etc.

Based on this definition, we can situate perceived affordances, feedforward and feedback within Norman's Stages of Action model (Norman, 1988), as shown in Figure 5.2. We would like to point out that Figure 5.2 is based on Norman's original illustration of the Stages of Action model (Norman, 1988), instead of the revised version (Norman, 2013b) that we used in Section 2.3.1. We do this for historical accuracy in relation to other definitions that have analysed the role of affordances and feedforward in relation to Norman's model (see Section 5.4).

Feedback bridges Norman's Gulf of Evaluation—the amount of effort users must exert to interpret the state of the system and to determine how well the expectations and intentions have been met. Recall that, when evaluating the state of the world, users go through the Stages of Evaluation shown on the right side of Figure 5.2 (see also Section 2.3.1). Feedforward, on the other hand, bridges Norman's Gulf of

2 Gibson (1977, 1979) originally defined affordances as the available action possibilities in the environment in relation to an actor, independent from whether they can be perceived or not. For further details on the differences, see Section 5.4.1.

Execution—the difference between the user's intentions and the allowable actions—by helping users decide what action to take based on that action's expected outcome. When users act on a certain goal, they go through the Stages of Execution seen on the left side of Figure 5.2. Perceived affordances also help to bridge Norman's Gulf of Execution, but they serve a different purpose: suggesting a particular action to users, such as pressing a button, or turning a knob.

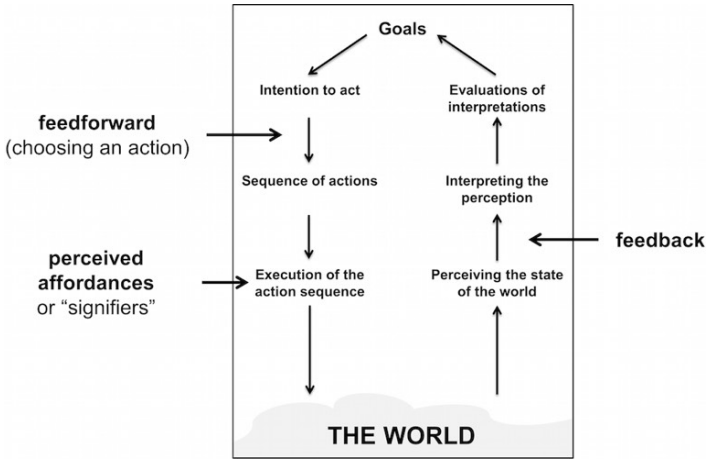


Figure 5.2: The role of perceived affordances (or signifiers: see Norman, 2008), feedforward, and feedback in Norman's Stages of Action model (image based on Norman, 1988).

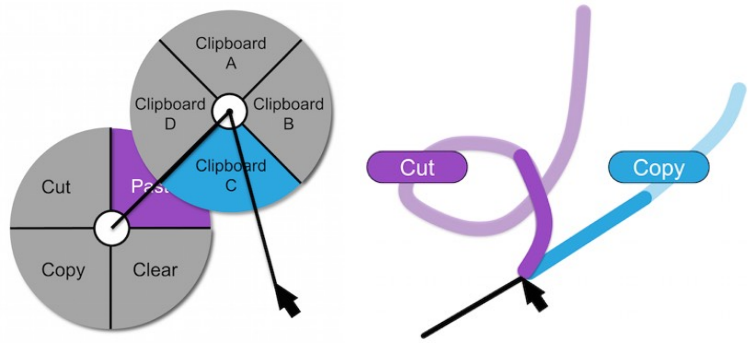
In the next sections, we will explore feedforward in depth. We start with current use of the term—which is at times inconsistent and thus illustrates the need for a more precise definition. Next, we discuss other definitions of feedforward to develop our own reframing of feedforward in which we further elaborate on its relation with perceived affordances and feedback. Finally, we give an overview of what the different definitions cover in terms of feedforward and analyse a few notable examples of feedforward.

5.3 USE OF FEEDFORWARD

Djajadiningrat et al. (2002) mostly focus on the importance of feedforward for tangible interaction. However, there have also been other application domains in which feedforward has been successfully applied, such as gestural interaction. As discussed earlier (see Section 3.2.1), a common problem of gestural interfaces is their lack of visibility: users lack awareness of the available gestures that are recognized by the system, and what these gestures do. Feedforward can help users in performing the correct gesture by showing the result of invoking gestures.

An early example of the use of feedforward in gestural interaction are Kurtenbach et al.'s marking menus (see also Section 3.3.4). Marking menus are circular menus that support gestural interaction and are intended to accommodate both novice and

expert users (Figure 5.3a). They allow a user to perform a menu selection by either popping-up a pie menu, or by making a straight mark in the direction of the desired menu item (without showing the menu). Bau and Mackay (2008) mention that the pie menu serves as a 'feedforward display' that helps novice users who hesitate when they are unsure of a gesture or command. When users become more experienced, they tend to use marks more, although they still look at the menu now and then to refresh their memory (Kurtenbach and Buxton, 1994). Note that marking menus show both what gestures are possible and what users can expect when they perform one of these gestures (e.g., 'Cut', 'Copy', 'Paste'). Bau and Mackay (2008) developed OctoPocus, a dynamic guide that combines on-screen feedforward and feedback to help users learn, remember and execute gestures. They state that "feedforward mechanisms provide information about a gesture's shape and its association with a particular command, prior to the execution or completion of the gesture." (Bau and Mackay, 2008). Like marking menus, OctoPocus takes advantage of possible hesitation by appearing after a "press and wait gesture". The set of possible gestures and associated commands are continuously updated while the user is performing a certain gesture, as illustrated in Figure 5.3b. Other examples of feedforward in gestural interaction are ShadowGuides (Freeman et al., 2009) and TouchGhosts (Vanacken et al., 2008), which extend Bau and Mackay's concept of dynamic guides to multi-touch gestures.



(a) Marking Menus (b) OctoPocus

Figure 5.3: Examples of feedforward in gestural interaction (images based on Kurtenbach et al., 1993 and Bau and Mackay, 2008).

Additionally, Bellotti and Edwards (2001) mention that feedforward is an important design principle for making context-aware systems intelligible. They mention that feedforward answers the question "What will happen if I do this?". Although Bellotti and Edwards adhere to Djajadiningrat et al.'s basic notion of feedforward—communicating the *purpose* or *result* of an action—they categorize it as a particular type of feedback (Bellotti and Edwards, 2001, pg. 203), which is not in line with Djajadiningrat et al.'s ideas. After all, Djajadiningrat et al. see feedforward and feedback as separate concepts, where one provides information *before* the action and

the other provides information *after* the action. Bellotti and Edwards (2001) also provide examples of feedforward in WIMP GUIs that are often taken for granted—but, as they argue, are necessary components of the interface that help users know what will happen when a certain action is performed—such as flashing insertion points; cursors, pointers and handles; window highlighting; and rubberbanding. The "What if?" questions by Lim and Dey (2010) can be seen as an example of the kind of feedforward that Bellotti and Edwards refer to.

By comparing Djajadiningrat et al.'s original definition of feedforward to the interpretations by Bellotti and Edwards (2001) and Bau and Mackay (2008), it is evident that the term feedforward has not always been used consistently. Different subcommunities within human–computer interaction seem to interpret the concept in a slightly different way. Moreover, there are only a handful of HCI textbooks that talk explicitly about feedforward. One of them is Dan Saffer's "Designing for Interaction" (Saffer, 2009), which refers to the definition by Djajadiningrat et al.. Saffer argues that designers should look out for opportunities to use feedforward—even though, as he mentions, "it is harder to design into products and services than feedback" (Saffer, 2009, pg. 133). After discussions with Don Norman about our paper on feedforward (Vermeulen et al., 2013b), Norman incorporated feedforward in his Stages of Action model in the 2013 revision of "The Design of Everyday Things" (Norman, 2013b; see further: Section 5.4.6).

5.4 FEEDFORWARD DEFINITIONS

In this section, we outline the differences between existing definitions of feedforward. This overview will provide a thorough review on the notion of feedforward and its relation to affordances. Table 5.1 lists different definitions and examples of feedforward alongside a number of important dimensions. We will discuss this table in more detail later (Section 5.5.4).

5.4.1 *Djajadiningrat: Going Beyond Affordances*

As previously mentioned, Djajadiningrat et al. (2002) have defined feedforward by contrasting it with feedback and affordances. Feedback is one of the most well-known design principles in interaction design, along with affordances, visibility, constraints and mappings. Feedback is a message about whether or not a goal was achieved or maintained (Saffer, 2009) and is typically used to inform the user that the system is responding, to indicate progress or to confirm navigation (Bellotti and Edwards, 2001). Djajadiningrat et al. (2002) state that feedforward, like feedback, returns information about the *result* of a process or activity. However, while feedback is communicated *during* or *after* the action, feedforward is information that is offered *before* the action takes place. Whereas feedback informs the user about the action that is carried out, feedforward informs the user about *what the result of their action will be*.

	Modality			Detail			Hierarchy	Time		
	Visual	Tactile	Aural	High	Average	Low	Nested	Static	Sequential	
									Discrete	Continuous
Definitions										
Djajadiningrat	✓	✓				✓		✓		
Wensveen	✓	✓	✓		✓	✓	✓	✓		
Gaver / McGrenere and Ho	✓	✓	✓				✓	✓	✓	
Hartson	✓						✓	✓		
Norman	✓					✓		✓		
Bau and Mackay	✓				✓					✓
Examples										
OctoPocus / ShadowGuides	✓				✓					✓
TouchGhosts	✓			✓					✓	
SpeakCup	✓					✓			✓	
Disney Pixar Cars 2 AppMATes	✓					✓	✓			✓
Tooltips	✓				✓			✓		
Tangible Video Editor	✓				✓		✓	✓	✓	
TempSticks	✓	✓				✓	✓	✓		

Table 5.1: Summary of the coverage of the feedforward definitions, their differences and an analysis of several feedforward examples in practice.

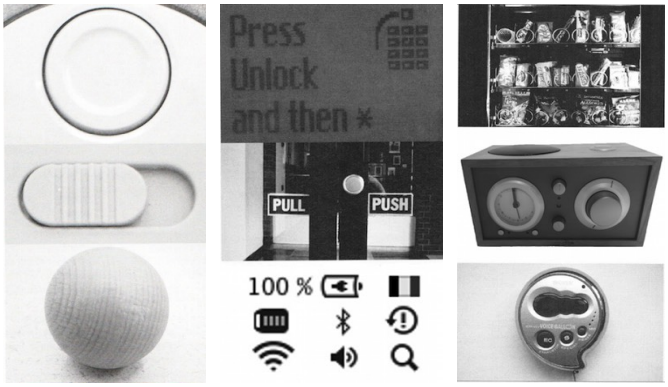
Next to feedforward, affordances also provide information to users before they carry out an action. Gibson (1977, 1979) defined affordances as:

> All 'action possibilities' latent in the environment, objectively measurable and independent of the individual's ability to recognize them, but always in relation to the actor and therefore dependent on their capabilities.

As introduced in the HCI literature by Norman (1988), perceived affordances (affordances that the user is made aware of through good design) essentially *invite the user to a particular action*. Affordances therefore *suggest* how one can interact with a product or system. Typical examples in HCI are buttons which *afford* pushing, knobs which *afford* turning, or sliders which *afford* moving up and down (or left and right, depending on the orientation). Affording the right actions has been widely regarded as a crucial aspect of usability. Even though perceived affordances are very useful, Djajadiningrat et al. (2002) argue that inviting the appropriate action is a prerequisite for feedforward, but it is not sufficient. They state that the essence of usability lies not in communicating the necessary action, but the *purpose* of an action.

5.4.2 *Wensveen: Inherent, Augmented & Functional Feedforward*

Wensveen et al. (2004) further elaborated on their previous definition (Djajadiningrat et al., 2002) and distinguish between three different types of feedforward, based on the 'form of information' that the user receives about their action: *inherent*, *augmented* and *functional* feedforward (Figure 5.4).



(a) Inherent (b) Augmented (c) Functional

Figure 5.4: Wensveen's three types of feedforward. Images from (Wensveen, 2005) reused with permission (Copyright © 2005 Stephan Wensveen).

Wensveen (2005) separates different types of feedforward in analogy with the way he distinguishes between inherent, augmented and functional feedback[3]. Wensveen explains the difference between these three types of feedback using the example of turning on a television set (Wensveen, 2005, pg. 158–160). *Inherent feedback* is information arising from acting on one of the action possibilities, from the movement itself—the click you hear when pressing the 'on' button, and the way it feels when you push it. *Augmented feedback* is information coming from an external source—the red LED that lights up to confirm that the TV is turned on. Finally, *functional feedback* is information generated by the system when performing its function—the picture shown on the TV screen and the sound generated by its speakers. Wensveen (2005) argues that—in a similar fashion—it is possible to distinguish between inherent, augmented and functional feedforward:

INHERENT FEEDFORWARD offers information related to the action possibilities of the product and appeals primarily to the *perceptual motor skills* of the person. Inherent feedforward communicates what kind of action is possible (e.g., pushing, sliding, rolling; see Figure 5.4a) and how this action can be carried out (the amount of force required, body parts, etc.). Wensveen (2005, pg. 161) states that "inherent feedforward can be viewed as a limited interpretation of Gibson's affordances" (Gibson, 1977).

AUGMENTED FEEDFORWARD is information *from an additional source* about the *action possibilities* of a product or system, or the *purpose* of these action possibilities (Wensveen, 2005, pg. 162). This type of feedforward appeals primarily to the *cognitive skills* of users. Figure 5.4b shows examples of augmented feedforward, such as on-screen messages indicating what to do (i.e., conveying the

3 The terms *inherent* and *augmented* feedback originated in the psychology of learning (see Laurillard, 1993).

action possibilities) and lexical or graphical *labels* communicating the purpose of the action possibility. As mentioned earlier, another example of augmented feedforward is the camera flash example shown in Figure 5.1 on page 73.

FUNCTIONAL FEEDFORWARD goes beyond the action possibilities and their specific purpose and instead informs the user about the *more general purpose of a product and its functional features* (Wensveen, 2005, pg. 162–163). A possible strategy for functional feedforward is making the functional parts visible—similar to Norman's notion of *visibility* (Norman, 1988). The candy vending machine in Figure 5.4c (top), for example, makes the available types of candy and the mechanism which delivers the products to the user clearly visible.

Note that by defining inherent feedforward as a limited interpretation of Gibson's affordances, Wensveen does not contradict his earlier work (Djajadiningrat et al., 2002)—where he does not yet differentiate between different types of feedforward. According to Wensveen (2010), when viewed holistically, feedforward as a whole—the combination of inherent, augmented and functional feedforward—still goes beyond affordances. Additionally, Wensveen (2010) clarified that the distinction between the three types of feedforward also relates to the kind of skills we primarily use to process that information: perceptual motor skills for inherent information, cognitive skills for augmented information, and the combination of emotional, cognitive and perceptual motor skills for functional information.

Even though the above-mentioned definitions distinguish between feedforward and affordances, there have been a number of frameworks for affordances (e.g., Gaver, 1991; Hartson, 2003; Kaptelinin and Nardi, 2012; McGrenere and Ho, 2000) that included aspects of feedforward without explicitly mentioning the term. In the next sections, we give an overview of these frameworks and explain how they relate to feedforward.

5.4.3 *Gaver: Technology Affordances*

In his paper titled "Technology Affordances", Gaver (1991) argues that affordances are not always single, independent entities, but can be related to one another. He describes two different relationships between affordances: *nesting* and *sequence*. Nested affordances are grouped in space, while sequential affordances are sequential in time (i.e., acting on an affordance leads to information indicating new affordances). Gaver states that affordances are not passively perceived, but *explored*. He also hints at the possibility of conveying affordances through different modalities (e.g., sound, tactile information).

Nested affordances, in particular, bear resemblance to feedforward. McGrenere and Ho (2000) have analysed Gaver's work, and clarify the concept of *nested* affordances with the example of a button. They state that users are not interested in clicking on a button for its own sake, but are interested in *invoking a certain function*. The function that will be invoked by a button is usually specified through its label or icon. They explain that here the affordance of 'button clickability' is nested within the affordance of 'function invokability'. McGrenere and Ho stress that it is

important to acknowledge that each of the levels of the affordance hierarchy may or may not *map to system functions*. Furthermore, they believe that affordances are not limited to physical aspects of the system (e.g., physical interaction with a mouse, keyboard or screen), as implied by Norman in his clarification of the use of the term affordances (Norman, 1999). They state that application software also provides possible actions. For example, a word processor affords writing and editing at a high level, but also actions such as clicking and scrolling or dragging and dropping. According to McGrenere and Ho (2000), the functions that the user can invoke are the (real) affordances in software (i.e., what the software 'affords'). Labels, icons and menus act as perceptible affordances that make these functions visible.

Given Gaver's extension of nested affordances, Wensveen's functional feedforward (Wensveen, 2005) could be seen as a perceptible affordance (Gaver, 1991) which conveys the general (top-level) function of a system—or what the system affords the user. This top-level affordance can be seen as the root of a hierarchy of affordances. As an example, in Figure 5.4c (bottom), the speech-bubble shape of the voice recorder conveys its general function to the user. However, to actually record speech, users will also have to be aware of the nested functions and affordances for these functions (e.g., the record button). One could argue that—through the concept of functional feedforward—Wensveen is implicitly suggesting that feedforward is nested as well, just like Gaver's affordances.

Gaver also introduced *sequential* affordances, which are only available at certain points in time. This is common in graphical user interfaces since these can be updated during usage. In contrast—unless we are dealing with shape-changing interfaces (Coelho and Zigelbaum, 2011)—physical objects usually have a static physical appearance and cannot update their form over time. Gaver (1991) notes that the information that specifies an affordance (e.g., a button on the screen), can be quickly updated as new affordances become available (e.g., adding a drop down menu when the button is clicked to allow the user to make a selection). Similarly, we argue that feedforward could only be made available at certain points in time or be updated during the user's action to provide new information. The examples of feedforward in gestural interaction that were discussed previously in Section 5.3 (marking menus and OctoPocus) can be seen as examples of *sequential feedforward*. Sequential feedforward in combination with feedback could further blur the difference between the two concepts. Feedback provided after performing an action might afterwards serve as feedforward for the action that logically follows the previous one.

5.4.4 *Kaptelinin and Nardi: Mediated Action & Affordances*

Kaptelinin and Nardi (2012) call for adopting a mediated-action perspective on technology affordances as an alternative for Gibson's ecological psychology perspective (Gibson, 1979). They differentiate between two types of affordances: *instrumental technology affordances* which are comprised of a *handling affordance* and an *effecter affordance*; and a set of *auxiliary technological affordances* such as maintenance, aggregation and learning affordances. We mainly focus on instrumental technology

affordances—in particular *effecter affordances*—here, as these appear to be quite similar to feedforward. Kaptelinin and Nardi (2012) explain the difference between handling and effecter affordances with the example of a knife. A knife consists of two distinct parts: the handle and the blade. The knife handle is used for holding the knife, while the blade is used to manipulate objects (e.g., an apple). They argue that this distinction also applies to digital technologies, and, more specifically, to user interface widgets. For example, the ability to drag the scroll box of a scroll bar is the handling affordance, while the ability to display a certain portion of the document in the window is the effecter affordance.

Like Djajadiningrat et al. (2002) and Wensveen (2005), Kaptelinin and Nardi (2012) distinguish between the *purpose of an action* (the effecter affordance) and the *action possibility* (the handling affordance). Indeed, they provide an example of a user interface dialog where: "handling affordances are clear but the outcomes of user actions (effecter affordances) are not immediately obvious. [...] The user can see that they can select the checkboxes and click the buttons, but the effects of these actions are not directly apparent." They state that users will be confused when handling and effecter affordances are not coupled tightly enough. Kaptelinin and Nardi's effecter affordances appear to be closely related to feedforward since they convey the outcome of a certain action. The idea of tightly integrated handling and effecter affordances seems to be similar to how perceived affordances and feedforward can be combined to communicate both the action possibilities and the expected outcomes of those actions.

5.4.5 *Hartson: Feedforward as a Cognitive Affordance*

Hartson (2003) further clarified the concept of affordances, extending Gaver's and McGrenere and Ho's work. He distinguishes between four types of affordances based on the role they play in supporting users during interaction:

COGNITIVE AFFORDANCES are considered to be an extension of Norman's perceived affordances (Norman, 1988), helping users with their cognitive actions. Hartson (2003) defines cognitive affordances as "a design feature that supports, facilitates, or enables *thinking and/or knowing about something*".

Example: A button label that helps users know what will happen if they click on it.

PHYSICAL AFFORDANCES help users with their physical actions, and match with Norman's real affordances (Norman, 1988)—or Gibson's affordances (Gibson, 1977, 1979). According to Hartson (2003), a physical affordance is "a design feature that helps, aids, supports, facilitates, or enables *physically doing something*".

Example: A button that is large enough so that users can click on it accurately.

SENSORY AFFORDANCES help users with their sensory actions (perceiving information). Hartson (2003) defines a sensory affordance as "a design feature that helps, aids, supports, facilitates, or enables the user in *sensing (e.g., seeing, hearing, feeling) something*". Sensory affordances play a supporting role for cognitive and physical affordances. Hartson thus explicitly separates sensing from understanding.

Example: A label font size large enough to read easily.

FUNCTIONAL AFFORDANCES are a design feature that help users *accomplish work*. It ties usage to usefulness, and is similar to McGrenere and Ho's idea of 'affordances in software'. Functional affordances add *purpose* to a physical affordance.

Example: The internal system ability to sort a series of numbers (e.g., invoked by users clicking on the 'Sort' button).

These four types of affordances are tightly coupled and work together to help users in their interaction. Physical affordances are associated with the 'operability' characteristics of user interface artefacts. Cognitive affordances are associated with the semantics or meaning of user interface artefacts. Sensory affordances have a supporting role, and are associated with the 'sense-ability' characteristics of user interface artefacts, especially of physical affordances and cognitive affordances. According to Hartson (2003), it is *design* that connects sensory affordances to physical and cognitive affordances, so that they can be seen, heard or felt to be used. Moreover, he notes that physical affordances carry a mandatory component of *utility* or *purpose*—the functional affordance—to which statements about physical affordances should refer.

Hartson's framework significantly broadens the scope of affordances, so that they also include both the notions of feedback and feedforward. A cognitive affordance is explained by the example of "a button label that helps users know what will happen when they click on it" (Hartson, 2003, pg. 323), which essentially reveals the purpose of this button. Of course, revealing the purpose of an action and informing users of "what will happen" corresponds to the original definition of feedforward by Djajadiningrat et al. (2002). Additionally, Hartson plugs his four types of affordances into Norman's Stages of Action model (Norman, 1988), as seen in Figure 5.5. The way he does this seems to suggest that *both feedback and feedforward are cognitive affordances*. Recall that we positioned feedforward and feedback in Norman's Stages of Action Model before (see Figure 5.2). Interestingly, Hartson identifies the need for cognitive (and sensory) affordances exactly where we situate feedforward and feedback. This is evident from Figure 5.6, which combines Figure 5.2 and Figure 5.5. Hartson later indeed confirmed that he sees feedback and feedforward as examples of cognitive affordances (Hartson, 2010, email communication).

Furthermore, we can also describe Wensveen's augmented, inherent and functional feedforward (Wensveen et al., 2004; Wensveen, 2005; see Section 5.4.2) in terms of Hartson's four types of affordances:

- *Inherent feedforward*: Wensveen (2005) sees inherent feedforward as a limited interpretation of Gibson's concept of affordance (Gibson, 1977, 1979). Wensveen notes that inherent feedforward communicates what kind of action is possible and how it can be carried out, regardless of its purpose (Wensveen, 2005, pg. 161). Norman made a difference between the presence of an action possibility, and the information that makes that action possibility visible, as is evident from the term *perceived affordance* (Norman, 1988). Wensveen's inherent feedforward deals with both the action possibility and the information that reveals it ("what action is possible"). According to Hartson (2003), the physical action possibility in itself, is a *physical affordance*—a design feature that helps users to physically do something. Revealing action possibilities happens through a combination of well-designed *cognitive* and *sensory* affordances. We can therefore conclude that inherent feedforward is the combination of a *cognitive* and *sensory affordance* that together reveal the underlying action possibility, or *physical affordance*.

- *Augmented feedforward*: this is a *cognitive affordance*, where lexical or graphical labels (e.g., words, icons, spoken words) communicate the result of the action. Hartson further separates the perception aspect through his notion of *sensory affordances*, which in this case would ensure that the labels are legible and sound is audible.

- *Functional feedforward*: informs the user about the more general purpose of a product and its functional features (Wensveen, 2005). As noted above, Wensveen's functional feedforward can be seen as a high-level nested affordance in Gaver's terminology. Even though Hartson does not explicitly state that his four kinds of affordances can be nested, we feel this is implied by several examples in his paper (Hartson, 2003). Wensveen's functional feedforward might thus be categorized as a *cognitive affordance* that makes the high-level functionality of the product—or its *functional affordance*—visible. Again, this cognitive affordance can be supported by a *sensory affordance*. For example, if the high-level functionality of the product is conveyed through its physical form, the discernability of that form can be seen as a supporting sensory affordance.

Note that Hartson's framework unites both Gaver's and McGrenere and Ho's work on affordances, and can be used to explain feedforward according to both Djajadiningrat et al.'s and Wensveen's definitions (Wensveen, 2005; Wensveen et al., 2004). While we started Section 5.4 with the assumption that feedforward goes beyond affordances, an analysis of Hartson's framework (Hartson, 2003) suggests that feedforward might just be a particular kind of affordance, i.e., a *cognitive* affordance. We will later use Hartson's framework to reframe feedforward and disambiguate it from affordances and feedback (Section 5.5). We conclude this section with an overview of aspects related to feedforward in Norman's work.

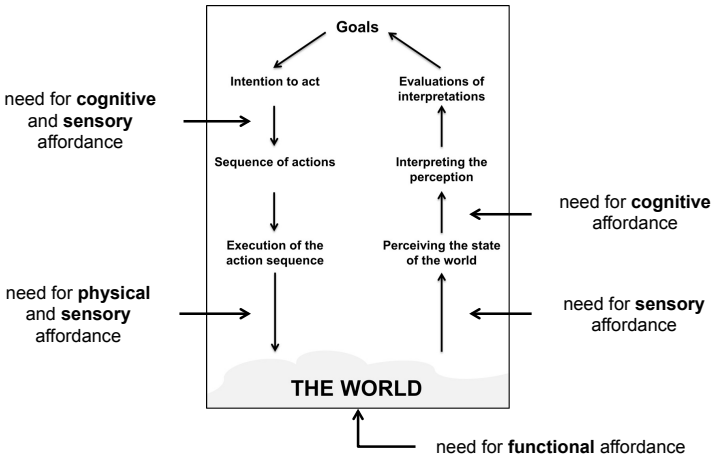


Figure 5.5: The need for physical, cognitive, sensory and functional affordances in Norman's Stages of Action model according to Hartson (image based on Hartson, 2003).

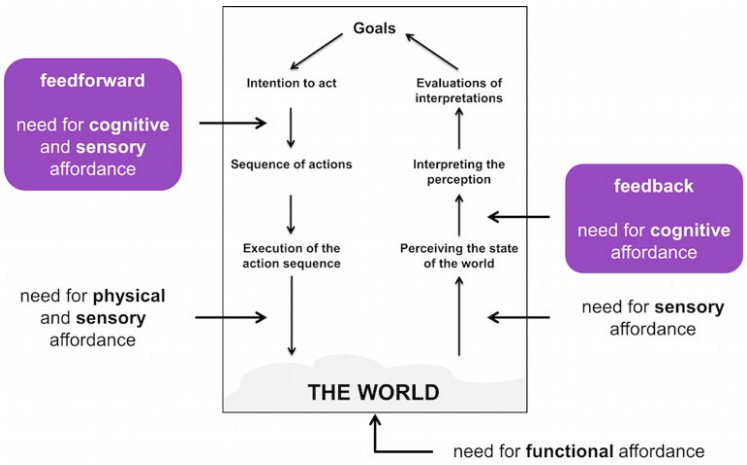


Figure 5.6: The result of combining Figure 5.2 and Figure 5.5, which suggests that both feedback and feedforward can be seen as cognitive affordances.

5.4.6 *Norman: Natural Mapping, Conceptual Models, Symbols and Constraints*

Recall that feedforward reveals the purpose of an action, which according to Djajadiningrat et al. (2002) is essential to create usable interfaces. However, in most of his examples, Norman (1988) always seems to imply a purpose for a physical affordance (e.g., a doorknob that can be turned *in order to* open the door). Hartson (2003, pg. 321) confirms this [emphasis ours]:

> In Norman's Design of Everyday Things world of non-computer devices, a purpose for a physical affordance is always implied. The doorknob is a cognitive and physical affordance for operating the door. The physical affordance offered by a doorknob does not mean merely that the doorknob can be grasped and turned. It means that the doorknob can be grasped and turned *in order to operate* (e.g., invoke the function or mechanism of opening) the door; the user is enabled to operate the door. In turn, the door itself is a functional affordance that, when invoked, allows passage. In this interaction design view, a physical affordance gives access to functionality, the *purpose* of the physical affordance used to access it.

Hartson (2003) notes that even though the addition of purpose to the description of a physical affordance is an obvious extension, it should be made explicit to avoid ambiguities in terminology. Moreover, when the interaction becomes more complex—as is certainly the case for context-aware systems—it also becomes more important for designers to explicitly communicate the purpose of an action.

Norman provides two ways to help users determine the purpose of a user interface artefact: *natural mappings* and a good *conceptual model* (Norman, 1988). As these mechanisms allow users to know what will happen when they perform an action, they could be seen as examples of feedforward. First of all, natural mappings allow users to determine the relationships between actions and results, between the controls and their effects and between the system state and what is visible by exploiting spatial relationships and temporal contiguity[4]. Common examples of natural mapping are light switches laid out in the same pattern as the lights in the room, or stove controls using the same layout as the burners they control. In both cases, the actionable elements are laid out in the same order as the artefacts in the physical world that they control. However, according to Djajadiningrat et al. (2004), mappings fall short when dealing with abstract data that has no physical counterpart. Secondly, a good conceptual model allows users to predict what will happen when they perform an action (see also Section 2.3.3). Norman mentions that this can be realized by exploiting consistency. He states that consistency in the presentation of operations and results and a coherent, consistent system image are essential to ensure that the user forms a correct conceptual model (Norman, 1988, pg. 53).

4 Temporal contiguity occurs when two stimuli are experienced close together in time, which allows an association to be formed. In the case of a light switch, the immediate feedback of the light turning on when we flip the switch, allows us to associate the switch with that light.

Two other important design principles proposed by Norman are *symbols* and *constraints* (Norman, 1988). Norman argues that these are not affordances and that wording in the label on a button, for example, is symbolic communication. Hartson (2003) agrees, but states that under his own definition, communication is exactly what makes good wording effective as a cognitive affordance. It helps the user in knowing (e.g., what to click on). In other words, Hartson (2003) sees symbols, constraints, and conventions as essential underlying mechanisms that make cognitive affordances—and therefore also feedforward and feedback—work. Hartson (2003) argues cognitive affordances play an enormously important role in interaction design. He states that they are key to answering Norman's question: "How do you know what to do?". Nevertheless, he acknowledges that the design of cognitive affordances can depend greatly on cultural conventions (or constraints) as a common base for communicating the meaning of cues from designer to user.

Feedforward in the Revised 2013 Edition of DOET

We had a few conversations over email with Don Norman about our paper on feedforward (Vermeulen et al., 2013b). Norman later included feedforward into his Stages of Action model (Norman, 2013b). In what follows, we briefly discuss Norman's view on feedforward.

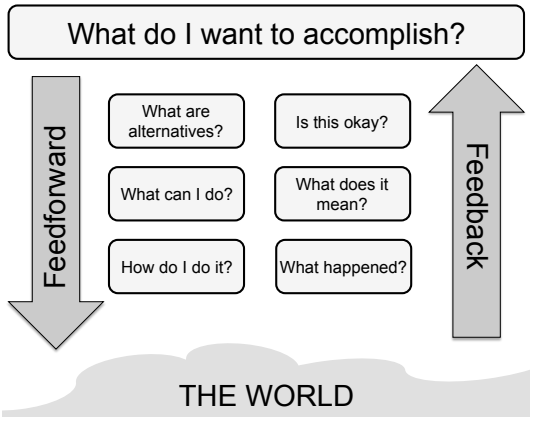


Figure 5.7: Norman's view on feedforward and feedback: feedforward answers questions of execution, while feedback answers questions of evaluation (image based on Norman, 2013b, pg. 71).

In the revised and expanded edition of "The Design of Everyday Things" (Norman, 2013b), Norman explains that he sees feedforward as an important aspect in bridging the gulf of execution, as shown in Figure 5.7. He mentions that feedforward is the information that helps answering questions of execution: "while feedback helps you know what happened, feedforward helps you know what you can do" (Norman, 2013b, pg. 72). According to Norman, feedforward is realized through the appropriate use of signifiers, constraints and mappings. Moreover, he states that the conceptual model also plays an important role. Note that this is in

line with our earlier arguments. Conceptual models indeed allow the user to predict what will happen. Likewise, mappings inform the user of the relationship between controls and their actions, and thus tell users what the result of their actions will be. Constraints limit the possible actions, and are therefore mainly useful for discoverability and preventing errors. Additionally, they can also help the user with knowing what will happen: natural mappings, for example, work by providing logical constraints (Norman, 2013b, pg. 130). Similarly, cultural constraints can give us insights into what we can expect (e.g., the colour red can be associated with dangerous actions).

Nevertheless, as is evident from Figure 5.7, Norman seems to have a somewhat broader view of feedforward that also includes *discoverability* and *how* users can perform certain actions. According to our own definition (see Section 5.5), however, action possibilities are conveyed through perceived affordances. We corresponded with Norman about this (Norman, 2013a, email), and received the following reply [quoted with permission]:

> *Feedforward is not about discoverability. In my definition and use of feedforward, it is specifically what should happen after performing an action. This builds up the expectations for the result, which can then be confirmed or disconfirmed.*
>
> *Hmm. I answered above without looking at my book. Figure 2.7 [note: reproduced here as Figure 5.7] certainly does imply a far greater range. Interesting. I wonder what I had in mind? Maybe I was thinking of the broader implications. My explanation, above, makes sense. But I do like the figure. Feedforward, in engineering control theory, is in the restricted sense I describe above. But maybe the expanded notion of Figure 2.7 makes a lot more sense for designing for people. I guess that is what I had in mind when I did the figure.*
>
> *Hmm. So right now, I would go back and change my first paragraph after the quote from your email: Yes, feedforward also answers questions about action possibilities and how to perform actions.*

When Norman mentions the broader implications of feedforward, he refers to the fact that informing users of what will happen can also help them to know what is possible, and how to perform actions. Norman's view on feedforward is therefore indeed broader than ours, which we will explain in detail in the next section.

5.5 REFRAMING FEEDFORWARD

In this section, we reframe feedforward informed by the above discussion of feedforward and related design principles such as affordances and feedback. We further clarify the differences between feedforward, (perceived) affordances and feedback based on Hartson's four types of affordances (Hartson, 2003). As Hartson not only incorporates Gibson's affordances (Gibson, 1977, 1979) and Norman's perceived affordances (Norman, 1988), but also broadened the scope of affordances to include both feedback and feedforward, we feel his framework is useful to reason about the

differences and interrelationships between these design principles. As discussed earlier, Hartson states that physical affordances carry a mandatory component of *utility* or *purpose*—the so-called *functional affordance*–to which statements about physical affordances should refer. Hartson's notion of conveying the *purpose* of an action is, indeed, nothing else than feedforward.

5.5.1 *Disambiguation: Affordances, Feedforward & Feedback*

As discussed before, Hartson situated his four types of affordances into Norman's Stages of Action Model (Norman, 1988). Remember that Hartson identified the need for cognitive (and sensory) affordances exactly where we positioned feedforward and feedback in Norman's Stages of Action model (see Figure 5.6), which suggests that both *feedback and feedforward are cognitive affordances*. Indeed, Hartson later confirmed this in personal email communication (Hartson, 2010). Nevertheless, even though it is a cognitive affordance, feedforward is also connected to the three other types of affordances.

Our new view on feedforward, feedback and perceived affordances is as follows:

PERCEIVED AFFORDANCES (Figure 5.8a) are cognitive affordances that are understandable through well-designed sensory affordances (e.g. a door's handle) and reveal a physical affordance (an action possibility), which is coupled to a functional affordance (the action's purpose). Perceived affordances occur before the user's action and *invite them to an appropriate action*.

FEEDFORWARD (Figure 5.8b) is a cognitive affordance that is understandable through a well-designed sensory affordance (such as a readable, descriptive label or an object's physical shape) and reveals the functional affordance (the system function) coupled to a physical affordance (the action possibility). Feedforward occurs before the user's action and *tells users what the result of their action will be*.

FEEDBACK (Figure 5.8c) is a cognitive affordance that is understandable through a well-designed sensory affordance (e.g. an informative message), and provides information about the result of a user's action, which is revealed to the user through the combination of physical and functional affordances. Feedback is provided during or after a user's action and *informs them about the result of performing their action*. Feedback can later turn into feedforward again (in combination with a perceived affordance) for another action that logically follows the previous one.

Figure 5.8 illustrates these definitions and shows how perceived affordances, feedforward and feedback relate to each other and are linked to Hartson's four types of affordances (Hartson, 2003). Both perceived affordances and feedforward tell users something about a particular action through a combination of a physical and functional affordances. Perceived affordances and feedforward essentially provide different information about the action that users have to perform to achieve their goals.

While *perceived affordances reveal the physical affordance*, which tells users that there is an physical action available and how to perform it, *feedforward reveals the functional affordance* and tells users what will happen when they perform that action.

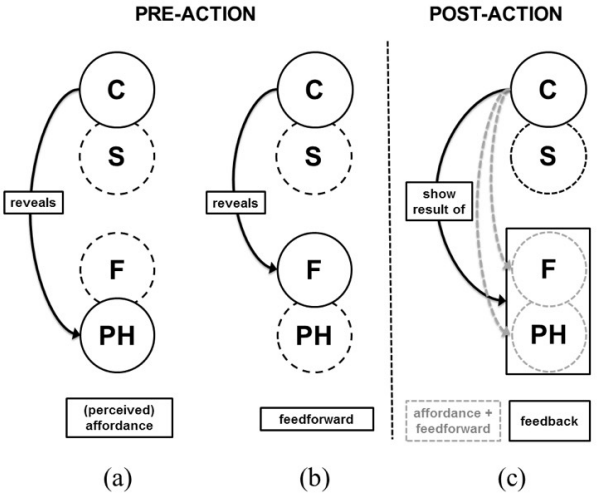


Figure 5.8: An overview of how (a) perceived affordances, (b) feedforward, and (c) feedback (c) can be explained using Hartson's four types of affordances. *C*, *S*, *F* and *PH* refer to Hartson's Cognitive, Sensory, Functional and Physical affordances respectively. In (c), the functional and physical affordances together constitute an action possibility. While perceived affordances and feedforward provide information before the user's action (*pre-action*), feedback occurs after the user's action.

An action here can be viewed at different levels of abstraction. In the provided examples, we mainly conceptualise an 'action' at an intermediate level of abstraction, in line with Hartson (2003). Although clicking a button can be broken down further in several precise motor movements (e.g., moving the mouse cursor to the button's position on the screen, pressing the mouse button and releasing it), we view this as a single physical action (with a corresponding physical affordance). Similarly, in Figure 5.1 on page 73, taking a picture is considered to be a single action. When users know how to perform elementary user interface actions, which in the case of WIMP interfaces would be actions like clicking, scrolling, or dragging, feedforward is mainly useful for designers (and users) at this intermediate level of abstraction. Nevertheless, feedforward could also assist users with lower level actions to accommodate users who are new to certain interaction styles. For example, feedforward in OctoPocus (Bau and Mackay, 2008) helps users perform gestures by answering the question "what happens if I make a gesture in this direction?".

Feedback provided after performing an action might afterwards again serve as feedforward for the action that logically follows the previous one (Figure 5.8c). Wensveen refers to feedback that turns into feedforward as *inherent traces of action*. In its simplest form, it is "nothing more than evidence for the user that he has acted on the action possibilities, as if it were a trace of the bygone action" (Wensveen

et al., 2004, pg. 183). An example of feedback turning into feedforward is a physical light switch. When users flip the switch, feedback consists of the changed position of the switch, and, of course, also the light that lights up—Wensveen's notion of functional *feedback* (Wensveen, 2005). However, the user's action also changed the possibilities for action, as the light cannot be turned on again, it can only be turned off. In essence, the feedback of the light and the state of the switch, becomes feedforward indicating that flipping the switch again will reverse the state of the light and thereby turn it off. Another example of feedback that turns into feedforward can be found in marking menus (Kurtenbach et al., 1993). Once a function in the marking menu is invoked, its label is changed to the corresponding inverse function. This inverse function label, at first, serves as feedback to indicate that the previous function has been invoked, and secondly, as feedforward for invoking the reverse function (thereby undoing the earlier action again).

5.5.2 *Hidden and False Feedforward*

Gaver (1991) also discerns between affordances—as in Gibson's original definition (Gibson, 1977, 1979)—and the perceptual information available about them, so-called *apparent* affordances. Apparent affordances correspond to what Norman defined as *perceived* affordances (Norman, 1988). Based on this distinction, Gaver introduces the concept of *false* and *hidden* affordances, where the apparent information about the affordance is either incorrect or missing.

Similar reasoning could be applied to feedforward. We introduce *false feedforward* and *hidden feedforward* and explain them using Hartson's framework (Figure 5.9). Feedforward is *false* when it conveys incorrect information about what system function the action performs. When feedforward is missing, it *hides* how the action is related to the system function, i.e., there is no information that reveals the purpose of the action. Although (usually) undesirable, false and hidden feedforward might be useful notions to consider in interaction design.

An example of *hidden feedforward* is an incomprehensible bank of light switches, where there is no information available about which switch controls which light[5]. Even though the switch clearly communicates that there is an action possibility, but there is no way of knowing what it will do. The only option for the user is often to try them all. Note that we do not have to put labels on the light switch to solve this problem. If Norman's concept of *natural mapping* is used to lay out switches in a similar spatial pattern as the room layout, then there is information that allows us to determine which switch controls which light. Another possibility was provided by Park et al. (2014). They integrated touch sensors in light switches to provide an additional touch state, similar to the 'hover' state in GUIs. When touching the light switch, the light that is controlled by that switch would light up, allowing the user to make the connection between both.

A very basic example of *false feedforward* is a button with an incorrect label. Although simple, this is an effective technique often employed by malicious software

5 For an example, see: Norman (2013b, Fig. 4.4.; pg. 136).

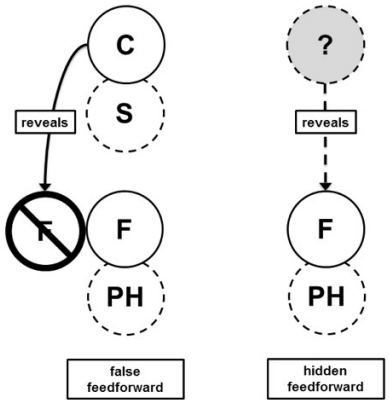


Figure 5.9: False and hidden feedforward. False feedforward provides incorrect information about the functional affordance, while hidden feedforward provides no information about the functional affordance that is coupled to the action.

to trick the user into invoking certain destructive actions. Figure 5.10 shows a UI dialog for so-called *scareware*—software that scares users about viruses on their computers and tricks them into installing malicious software that actually infects their computers (BBC News, 2009). Instead of removing malware from the user's computer, clicking the 'Fix Now!' button will actually install it—an example of providing incorrect information about the functional affordance.

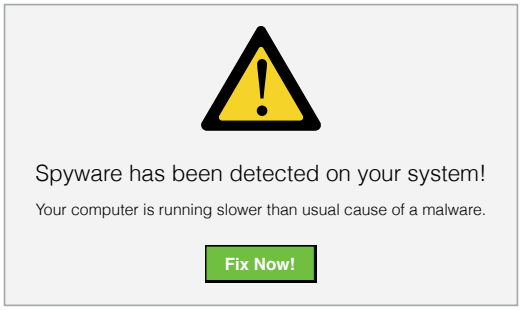


Figure 5.10: An example of false feedforward: so-called 'scareware' that tricks users into installing malware, although it actually advertises to clean the user's computer.

False or hidden feedforward (and affordances) relate to the notion of so-called *deceptive user interfaces*. Designers that create interfaces that deliberately deceive can either do so to benefit the user (Adar et al., 2013), or to trick them into doing things they would otherwise not do (Brignull, 2011). An example of benevolent deception is a progress bar that smooths the actual progress of the system, making the system 'feel faster' (Harrison et al., 2010). On the other hand, Brignull (2011) describes several *dark patterns* with less admirable goals. These patterns are used by e-commerce websites to deliberately trick users, for example into signing up for

monthly paying memberships when they were just interested in buying a single item.

5.5.3 *Nested and Sequential Feedforward*

Gaver (1991) proposed the idea of *sequential* affordances and *nested* affordances for complex actions. Similarly, we argue that feedforward can also be nested or sequential, which we illustrate next with a couple of examples. In his discussion of sequential affordances, Gaver further mentions that affordances can be conveyed through multiple modalities (e.g., visual, tactile, auditory information). Just like affordances, feedforward can also be provided using different modalities, as confirmed by Wensveen (2005) and Djajadiningrat et al. (2002). However, some modalities (e.g., tactile) will be better suited to exploratory actions as they cannot be perceived through what Gaver (1991) calls "relatively passive perception".

As previously discussed, functional feedforward (Wensveen, 2005) conveys the general purpose of a system. A common way to inform the user of a product's general purpose is through its form (e.g., the voice recorder in Figure 5.4c). This top-level function can be seen as the root of a *nested feedforward hierarchy*, and can be combined with feedforward that is provided for lower-level or *nested* functionality (e.g., the different buttons on the voice recorder).

Notable examples of *sequential feedforward* are systems that use feedforward to make gestural interfaces easier to use, such as OctoPocus (Bau and Mackay, 2008), ShadowGuides (Freeman et al., 2009). OctoPocus and ShadowGuides *continuously issue dynamic feedforward and gradual feedback during input*. While performing a gesture, users are provided with information about their current set of possible gestures (i.e., their action possibilities) and the expected result of performing those gestures, together with feedback about how well the current gesture has been recognized. Here, feedforward is the information that conveys the result of performing a particular gesture. Note that the level of detail of feedforward may vary. OctoPocus and ShadowGuides, for example, just use simple labels (e.g., 'Cut', 'Copy') to tell users what a certain gesture will do, as shown in Figure 5.3b. Other gesture learning systems, such as TouchGhosts (Vanacken et al., 2008), provide detailed animations of the effect of performing a gesture, such as resizing or removing an object. Feedforward could also be made available at *discrete points* in time, instead of being updated continuously. Updating feedforward over time is easily feasible in software, compared to physical interfaces. Sequential feedforward in combination with feedback could further blur the difference between the two concepts since feedback might afterwards serve as feedforward for the user's next actions.

5.5.4 *Retrospect: Definitions and Examples*

Table 5.1 on page 79 shows which aspects of feedforward are covered by the different definitions and clarifies how feedforward is used in a number of notable examples. The table provides several dimensions:

- the *modality* that is used to convey feedforward
- the *level of detail* with which feedforward is provided
- whether there is a nested feedforward *hierarchy*
- how feedforward is provided over *time*

It is clear from the definitions by Djajadiningrat et al. (2002) and Wensveen (2005), that feedforward can be provided using multiple *modalities*. However, designers mostly rely on visual information to convey feedforward, apart from a few exceptions, such as the tangible programmable heating controller TempSticks (Djajadiningrat et al., 2002).

Bau and Mackay (2008) have introduced the *level of detail* as a useful criterion for classifying feedforward mechanisms. Usually, feedforward is provided in a low to average amount of detail. An example of a low amount of detail would be a simple label on a button. However, there are situations in which feedforward can be provided with lots of details, for example when it is important that users are reassured by exactly knowing the outcome of a certain action (e.g., when controlling a smart home). The TouchGhosts technique for learning multi-touch gestures (Vanacken et al., 2008) is an example of feedforward that is provided at a high-level of detail, as it essentially provides a live preview of what happens when performing a gesture.

Feedforward can be nested in a *hierarchy*. Figure 5.11 shows two examples that use nested feedforward: the Disney AppMATes iPad game[6] (YouTube ID: VaNzbCtxtcY) and the Tangible Video Editor (Zigelbaum et al., 2007) (YouTube ID: _auBtFb1WmE). Nested feedforward tends to rely on the object's shape to convey its general function (i.e., functional feedforward), combined with lower-level types of feedforward information. Disney's AppMATes is a children's toy which uses tangible toy cars that can be used on a tablet. In this case, the shape of the toy car serves as high-level feedforward that explains its general purpose. When children pick the toy car up and place it on the screen, the display will show small halos underneath the car representing its head lights. This acts as additional feedforward information indicating that the car and display are linked. Children can move the car on the display to play a racing game. In case of the Tangible Video Editor, the shape of the different building blocks indicate their function (e.g., a movie clip, a transition, etc.).

Finally, feedforward can be either static or updated over *time*, similar to Gaver's idea of sequential affordances (Gaver, 1991). An example of static feedforward would be a fixed label or the (static) shape of an object. Examples of sequential feedforward are the OctoPocus (Bau and Mackay, 2008) and ShadowGuides (Freeman et al., 2009) dynamic guides, the Tangible Video Editor and SpeakCup (Zigelbaum et al., 2008). When users connect different building blocks together in the Tangible Video Editor, this arrangement of blocks provides new information that serves as feedforward for knowing how the movie clips will be combined together. SpeakCup

6 Disney AppMATes: `http://www.appmatestoys.com/`

(a) Disney AppMATes (b) Tangible Video Editor

Figure 5.11: Two examples of nested feedforward: Disney AppMATes and The Tangible Video Editor (image sources: YouTube).

is a digital voice recorder that is shaped like a small rubber disc with holes in its centre. When the holes are pressed in, forming a small cup, SpeakCup absorbs sound. When the holes are pressed out, the stored sounds are released. SpeakCup uses its shape to communicate to users what they can expect. As this shape changes over time when the disc is pressed in or out, SpeakCup is another example of sequential feedforward.

5.6 CASE STUDY: THE FEEDFORWARD TORCH

5.6.1 *Introduction*

To conclude this chapter, we illustrate how feedforward can be used to help users interact with complex systems, appliances or controls. We introduce the *Feedforward Torch*, a mobile device that can be used to project information onto objects in the user's physical environment. The Feedforward Torch consists of a smartphone, mobile pico projector and a laser pointer enclosed in a plastic case. There is a hole at the bottom of the case that exposes the button of the laser pointer, which can be used to aim at objects. Using a torch light metaphor (Rukzio et al., 2012), the Feedforward Torch can be pointed at existing objects to reveal additional information about them. In particular, we focus on providing feedforward to reveal the purpose of certain controls or objects. The Feedforward Torch can, for example, be pointed at a light switch to know which light it controls, as shown in Figure 5.12.

In developing the idea for the Feedforward Torch, we were inspired by our earlier explorations into providing in-situ information using mobile projectors (see Section 4.3.3). Note that the Feedforward Torch is not a fully functional system. We use the Wizard of Oz technique (Kelley, 1984) to decide what information is shown and when it is shown. There is no location tracking or object recognition involved; what is projected is fully controlled by the *wizard*.

Our main objective with the Feedforward Torch was to explore whether users would appreciate in-place information about what would happen when interacting with objects and appliances in their everyday environments. We provided feedforward in a number of ways: visualizations, animations, textual explanations and

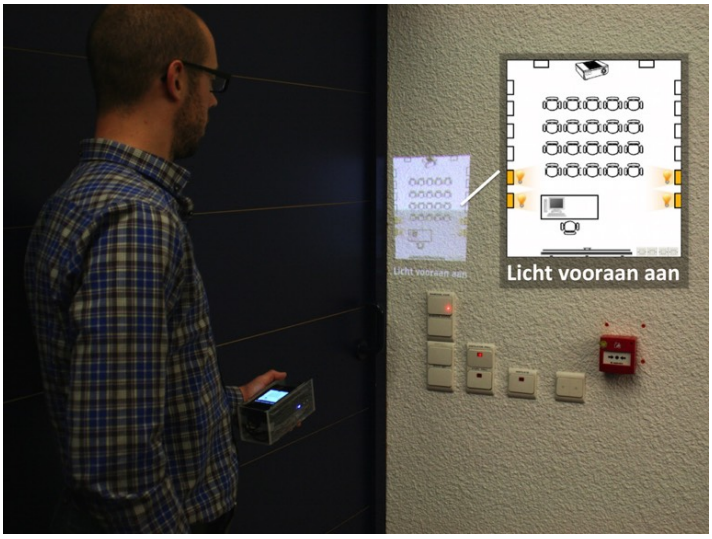


Figure 5.12: Using the Feedforward Torch to understand a bank of light switches. First, the user aims for the right object by pressing a button on the bottom of the device that activates the laser pointer. When he points at a particular light switch, a visualization of the room is projected that shows which light(s) will turn on when flipping that switch (call-out).

combinations thereof. Animations are used to better convey the effect an action will have, such as when the effect of an action happens after a delay, when it occurs in a different location or is invisible to the user, or when it takes place over a longer period of time (see Section 5.6.4).

5.6.2 *The Prototype: Hardware and Functionality*

As mentioned before, the Feedforward Torch consists of the combination of a smartphone, mobile projector, and laser pointer. To allow for one-handed interaction, these three components are enclosed in a custom-designed acrylic case, as shown in Figure 5.13 (left). At the top of the case, there is a Samsung Galaxy S Android smartphone with TV-out support. It is connected to a MicroVision SHOWWX$^+$ laser pico projector, which is located in the middle of the case. Below the pico projector, there is a red laser pointer that is taped to the case to keep it at a fixed position. The case has a small hole at the bottom to expose the laser pointer's button. Finally, Figure 5.13 (right) shows the Wizard of Oz controller, a second Android smartphone that runs a specific application to control what information is projected using the Feedforward Torch. The two devices are connected to each other over 3G.

Figure 5.14 (right) shows the Wizard of Oz controller application. The controller interface relies on configuration files that describe the different locations, objects of interest at those locations, and available feedforward information for these objects.

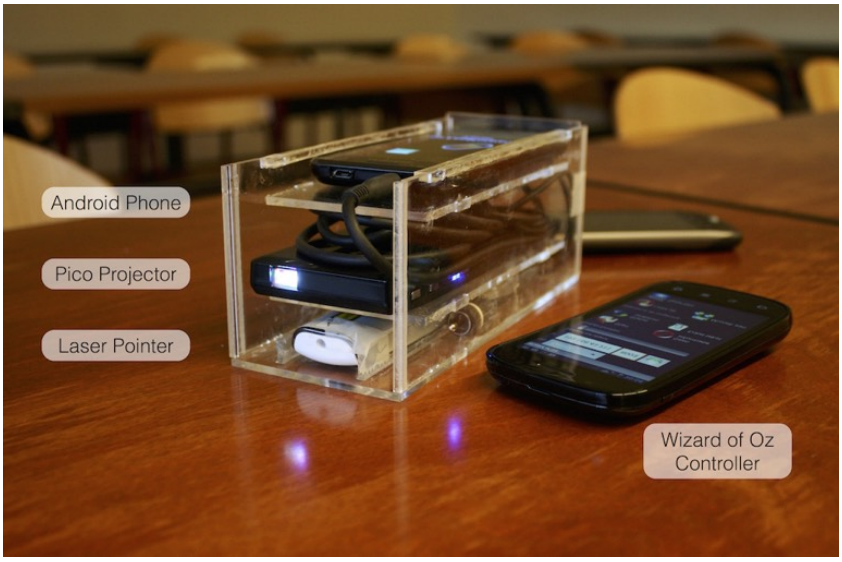


Figure 5.13: The Feedforward Torch encloses an Android smartphone, pico projector and a laser pointer in a plastic case (left). It is connected to a Wizard of Oz controller that runs on another Android smartphone (right).

The Feedforward Torch supports textual explanations, visuals, animations and combinations thereof.

The wizard stands in the back with the Wizard of Oz controller observing the participant. She can turn off the projection when the user is not pointing at an object, or select a specific room and a particular visualization from a list of preconfigured systems that were explored in the preliminary user study (Section 5.6.4). Additionally, a visualization can be shown that tells the user that there is no information available about the selected object. Given a specific visualization that is currently being shown, the wizard can easily show the visualization for the opposite action (e.g., replacing "lights on" with "lights off" when the user flips a light switch). The wizard can also set up the Feedforward Torch to only show textual explanations. Finally, the Feedforward Torch supports a display-only mode, where the projection functionality is disabled. Note that in this case the user can still point at an object of interest using the laser pointer, but information will only be shown on the smartphone display.

5.6.3 *Related Work*

The possibility of augmenting physical environments using mobile projectors was first demonstrated by Raskar et al. (2003) with their iLamps project. Earlier work by Pinhanez (2001) focused on steerable, ceiling-mounted projectors. Later, Raskar et al. (2004) extended these mobile projectors with RFID readers and photosensing capabilities to identify the physical objects that were being augmented. In the last

Figure 5.14: The Wizard of Oz controller application (right) provides controls for showing specific visualizations categorized in different rooms (see the drop-down menu at the top). The wizard can also turn off the projection, indicate that no information is available, or show a visualization for the opposite user action.

decade, advances in hardware have enabled compact prototypes that can be embedded into smartphones, and different interaction possibilities have emerged (Rukzio et al., 2012). According to the classification of applications for personal projectors by Rukzio et al. (2012), the Feedforward Torch would be in the augmented reality category. Indeed, the Feedforward Torch can be seen as an application of augmented reality (Azuma, 1997), as it essentially superimposes virtual information (i.e., feedforward) upon the real world.

While the Feedforward Torch is inspired by existing work on portable projectors, its contribution is not in producing high-quality projected graphics on (potentially curved) surfaces (Raskar et al., 2003), sophisticated object recognition or tracking technologies (Molyneaux et al., 2007), or interaction techniques (Cao et al., 2007; Harrison et al., 2011; Molyneaux et al., 2012). Instead, we aimed to explore whether users would appreciate the availability of in-place feedforward about complex devices in their everyday environments. Providing in-place guidance and instructions has been one of the pinnacle applications of augmented reality (Feiner et al., 1993). Researchers have explored different types of guidance using personal projectors, such as wayfinding (Wecker et al., 2011; Rasakatla and Krishna, 2013) showing warning labels on physical objects (Molyneaux et al., 2007), and augmenting on-screen instructions (Rosenthal et al., 2010). However, in contrast with these systems, we specifically focus on providing feedforward.

In our own work (Vermeulen et al., 2009a; Section 6.3), we have used steerable projectors to overlay an intelligent environment with real-time visualizations of ac-

tions occurring in this environment (e.g., lights that are turned on or off based on the presence of someone in the room). The Feedforward Torch serves a similar goal, but focuses more specifically on revealing the purpose of controls and requires less infrastructure. Moreover, with the Feedforward Torch, users are able to decide about what object they require information about and when they need it. Although the scenarios we describe for the Feedforward Torch do not involve context-aware systems, we feel the technique would be useful in those situations. However, when dealing with implicit input or autonomous system actions, the Feedforward Torch would have to switch to a system-driven approach where it proactively provides feedforward to show users what will happen. In Chapter 6, we present a general technique for increasing awareness of system actions and providing users with opportunities to intervene: *slow-motion feedback*.

5.6.4 *User Study*

We conducted a small-scale user study to assess (1) whether the Feedforward Torch helps users to better understand how to work with complex systems and (2) whether visualizations and animations are preferred over textual explanations. During the study, an experimenter (the wizard) was standing behind the user to observe their actions and control the Feedforward Torch. Participants were not told that the device was operated by the experimenter.

5.6.4.1 *Participants and Tasks*

The Feedforward Torch was used by 7 participants (5 male, 2 female; 4 without and 3 with a technical background; ages ranging from 28 to 40, mean = 32.14) in three different scenarios, as shown in Figure 5.15:

- Scenario (a): "The Television and the Timer". Participants were asked to turn on the TV in a classroom, and had to work around the timer that controls the TV. The timer is used to provide power to the TV and VCR, and automatically turns both devices off again after two hours.
- Scenario (b): "The PingPing NFC Terminal". Participants were instructed to recharge their contactless PingPing[7] payment card for the amount of 10 EUR. To do so, they had to use both their debit bank card as well as their PingPing NFC card.
- Scenario (c): "The Lecture Hall". In this scenario, the objective was to prepare the lecture hall for a presentation. This means the projector should be turned on, the projection screen should be lowered, and the lights should be dimmed.

7 https://www.pingping.be/

(a) The Television and the Timer (b) The PingPing NFC terminal (c) The Lecture Hall

Figure 5.15: The three scenarios participants encountered in the study of the Feedforward Torch.

5.6.4.2 *Procedure*

After a five-minute introduction of the Feedforward Torch, participants were given a specific goal they had to achieve in each of the three scenarios (e.g., turning on the TV). None of the participants were familiar with the different devices used in these scenarios. Each scenario took place in a different location around the university campus.

Before participants started to explore how to complete the predefined goal, they were asked to describe to the observers how they thought the devices in the scenarios should be used for this purpose. Their assessment of the system was only based on its appearance and labels or signs already present in the physical space. Next, the participant was asked to use the Feedforward Torch to complete the assigned task. Participants were alternately shown visualizations versus text-only information in the different scenarios in randomized order. The Feedforward Torch was configured to simultaneously project information and also show it on the smartphone display. Participants were told to try out both display methods during the study, and were later asked about their preferences.

After all three tasks had been performed, we conducted semi-structured interviews in which we inquired participants about the usefulness of the Feedforward Torch, and their preferences with respect to visualizations versus textual explanations. Moreover, they were asked in which situations mobile projection was preferred over showing information only on the phone display. The questionnaires used for the study can be found in Appendix B.2.

5.6.4.3 *Observations*

USEFUL FOR DEALING WITH COMPLEX SITUATIONS All participants were able to complete the tasks using the Feedforward Torch. When asked about its usefulness, all participants mentioned they found the Feedforward Torch useful to guide them through complex situations. Several participants mentioned they would have been unable to complete the three scenarios without the Feedforward Torch or additional help from the experimenters. Two participants stated that the system would

have come in handy when using the underground in a large city such as Paris or London. One participant said: "When I had to use the London Underground for the first time, it would have been useful to have a device like the Feedforward Torch to help me figure out how to use the ticketing machine. Now, I had to observe other passengers first before I knew how the system worked and what I had to do."

VISUALIZATIONS PREFERRED OVER TEXTUAL EXPLANATIONS Participants had a strong preference for visualizations over textual explanations in the encountered scenarios, as they considered reading text to be more time-consuming. However, a number of users suggested providing detailed textual descriptions as an secondary source of information to complement the existing visualizations.

ANIMATIONS USEFUL IN COMPLEX SITUATIONS As we expected, participants appreciated the use of animations, especially when the result of a certain action would happen over time or outside the user's periphery. In scenario (c), we showed for example an animation of the projection screen coming down when the participants selected the corresponding button.

MIXED FEELINGS ABOUT MOBILE PROJECTION The study revealed both advantages and disadvantages of mobile projection technology. Participants liked the fact that information was overlaid on the physical environment, so they did not have to switch between the phone display and the device they had to operate. One of the advantages of mobile projection brought up during the semi-structured interviews was the fact that groups of people could explore the projection together. However, participants were also concerned about potential privacy problems, in line with earlier findings by Raskar et al. (2003) and Rukzio and Holleis (2010). A disadvantage was the difficulty of using mobile projection in low light conditions and on curved surfaces, which are common issues (Rukzio et al., 2012). There was no clear preference for mobile projection, although we do expect hardware advancements to further improve the user experience. Based on our observations from the study, we do feel that using an augmented reality approach for showing feedforward information is valuable, due to the capability for displaying information exactly where it is needed. Although our prototype used a mobile projector for this purpose, we can imagine that other augmented reality technologies such as Google Glass[8]) could be used as well.

5.6.5 *Discussion*

With the Feedforward Torch, we explored how feedforward could help users to interact with devices or objects in complex situations. Although we did not explicitly evaluate the device in context-aware systems or environments, we feel that our findings could apply to those situations as well. For example, animations could be used in a smart home if the user's actions would take effect in another location or

8 http://www.google.com/glass/

after a certain period of time (e.g., when changing the home's energy and temperature control settings). To make this possible, the Feedforward Torch should not only be able to automatically recognize the object the user is pointing at, but also communicate with that object and retrieve a model of its behaviour. In Chapter 7, we describe how a context-aware system's behaviour model can be analysed to provide automatic explanations of 'why' the system acted in a certain way. There is potential in using similar techniques to automatically provide feedforward as well (i.e., answering 'what will happen').

One of the key features of the Feedforward Torch is its ability to provide on-demand and in-situ information. Analysing the Feedforward Torch using the design space introduced in Chapter 3, it can be categorized as a *user-driven*, *embedded* technique (see Table 3.7 on page 48). With respect to the timing dimension, the Feedforward Torch provides information *before* the user's or system's action. In Section 6.3, we introduce another embedded technique, one that, unlike the Feedforward Torch, is system-driven and mostly provides information during execution of actions.

5.7 CONCLUSIONS

In this chapter, we introduced *feedforward*, an important design principle to bridge Norman's gulf of execution. Like affordances, feedforward is a concept that may be interpreted in different ways, and may not be easy to define. However, although feedforward is often used by designers—as is evident from the examples discussed in this chapter—they tend to do so unknowingly and mostly based on their existing experiences and skills. We addressed this problem by reframing feedforward in terms of Hartson's four types of affordances, and disambiguating it from the related design principles feedback and perceived affordances. In addition, we identified four new classes of feedforward: hidden, false, sequential and nested feedforward, and analysed several existing examples of feedforward. Finally, we presented the *Feedforward Torch* to illustrate how providing feedforward can help users bridge the gulf of execution when interacting with complex devices or controls. In Chapter 6, we describe *slow-motion feedback*, a technique to provide intelligibility and control *during* actions.

6

SLOW-MOTION FEEDBACK

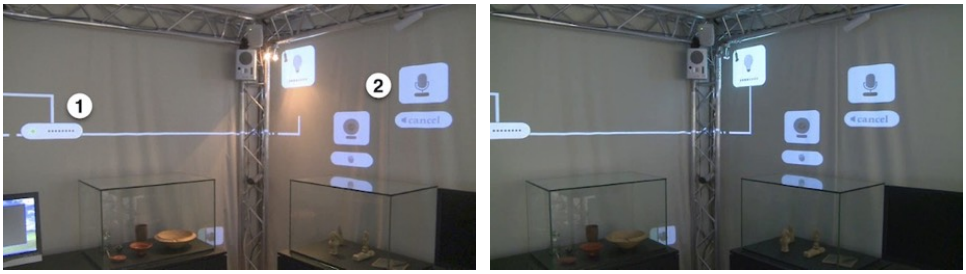
6.1 INTRODUCTION

When context-aware systems rely on implicit input and act autonomously, there is considerable potential for miscommunication between the user and the system. As discussed earlier (see Sections 2.3.2 and 2.3.4), interaction might take place beneath the user's radar (Bellotti et al., 2002), leading to unintended actions, undesirable results and difficulties in detecting or correcting mistakes (Ju et al., 2008). We have argued before that it is important for users to be able to intervene when the system makes a mistake (see Figure 2.6 in Section 2.3.2). Yet, before users can do so, they must first understand what the system is doing.

Understanding what the system is doing can be complicated by the misalignment between the system's time frame and that of its users, as argued by Bellotti et al. (2002). After all, computers can take action in a fraction of a second, much faster than we humans are able to notice. This only further complicates the challenge of providing timely feedback, as the user's implicit input could potentially trigger thousands of system actions before being aware of having interacted with the system. By the time the user realizes out what is going on, it might be too late to take corrective action.

In this chapter, we introduce *slow-motion feedback* (Vermeulen et al., 2014), a design concept that can be used to address these issues. Just as we would speak slowly when we explain something to a small child, computer systems could *slow down* when taking actions on the user's behalf and provide intermediate feedback to make sure that the user understands what is happening. Slow-motion feedback is a way to provide users with sufficient time to (i) *notice* what the system is doing and provides them with the opportunity to (ii) *intervene* if necessary.

To illustrate the idea of slow-motion feedback, we show an example in Figure 6.1. In this case, the user is in a context-aware museum environment that reacts to her movements and actions, such as showing additional information when she approaches artefacts in the museum. The environment's context-aware behaviour is driven by a set of context rules. One of those rules specifies that when movies are being shown, the lights in the museum will be dimmed to increase viewing comfort. In Figure 6.1a, a movie just started playing, which will thus result in the lights being dimmed. To make the user aware of this system action, lines are projected on the walls of the museum that visualize the different rules that are being executed. The lines connect different sensors and devices together to convey how they interact (Vermeulen et al., 2009a). In this case, an animated line moves from where the movie is playing and ends up at each of the four lights in the room. There is an annotation on the line that indicates that the lights will be turned off, as shown



(a) Before (b) After

Figure 6.1: An application of slow-motion feedback. Animations show that the system is about to dim the lights (a). The system's action is *slowed down* to allow users to notice what is happening, and provide sufficient time to intervene, if necessary. The lights are only dimmed when the animating line reaches them (b).

in Figure 6.1a(1). It is important to note that the lights are only dimmed *when* the projected lines reach them. The system's action is effectively delayed or *slowed down* to give users sufficient time to intervene. Moreover, the visualizations provide users with intermediate feedback and make them aware of what is happening. As shown in Figure 6.1a(2), there is a 'cancel' command available to stop the system action from being executed.

Slow-motion feedback essentially manipulates the time frame in which the system executes actions to realign it with the time frame of the user (Bellotti et al., 2002). Another example in which an action by the system is 'slowed down', is Gmail's 'undo send' feature (Figure 6.2). This feature provides users with a configurable 5- to 30-second window to undo sending out an email (which would technically be impossible to do). While Gmail shows feedback to the user informing them about the sent email, the actual sending of the email is delayed so that users have a chance to undo this action in progress. The email is sent after the specified time-out, unless the user clicks the 'Undo' button. This shows the wider applicability of slow-motion feedback, which is not about implicit interaction or autonomous actions here, but about providing an additional safety measure for users.

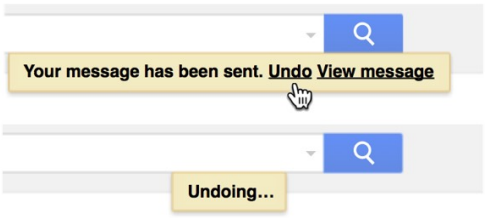


Figure 6.2: Another example of slowing down the system action: providing a specific time window during which sent emails can be 'undone' (source: Gmail).

Next, we introduce a design space to reason about the time at which feedback is provided, and use this framework to better define slow-motion feedback. Next, we delve deeper into the example shown in Figure 6.1, and discuss how we applied slow-motion feedback to provide real-time, in-place feedback using projected visualizations in a context-aware environment. Finally, we position other notable applications of slow-motion feedback in the design space, and end with a discussion of the potential of this technique for improving awareness and providing opportunities for control in implicit interaction.

6.2 DESIGN SPACE FOR THE TIMING OF FEEDBACK

6.2.1 *Introduction*

In order to define slow-motion feedback, we present a design space that allows us to define the different possibilities for how and when information about the result of an action can be provided. The design space consists of two dimensions: the time at which information is provided about the result of an action, and the level of detail of that information (Figure 6.3).

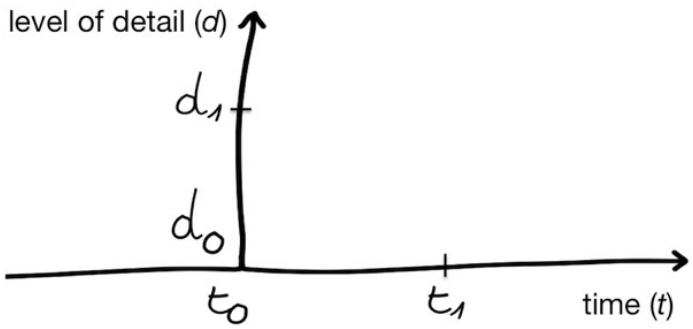


Figure 6.3: The design space for when and how information about the result of an action can be provided. These axes (*time* and *level of detail*) also apply to the rest of the figures in this chapter.

We can plot different feedback strategies in this space, defined by the following features:

- Time (t): The time dimension represents the time at which events occur, such as the start of an action, or when information is shown to the user. It consists of all moments in time that are relevant to the interaction. We define two key moments: at time t_0, the action is started (either by the user or the system); and at time t_1, the action has been completed by the system.
- Level of detail (d): This dimension represents how much information the user receives about the result of their action. While this dimension is hard to quantify and tends to have discrete levels, we define two important values for this

dimension: d_0 and d_1. The level d_0 represents the situation in which the user does not receive any information about the result of their action. At level d_1, the user receives fully detailed information about the result of the action.

- Origin and shape of the curve: We define the origin of the graph as (t_0, d_0); when the action starts and no information is provided yet. The curve's shape illustrates how information is revealed (in discrete steps or continuously), when and how much information is provided, and the amount of time the user has to intervene.

From these definitions, we can then derive a number of key regions, as shown in Figure 6.4:

- $t < t_0$: the time period *before* the action
- $t_0 \leqslant t \leqslant t_1$: the time period *during* the action
- $t > t_1$: the time period *after* the action
- $t_1 - t_0$: the amount of time available to the user to intervene, e.g., to cancel an unwanted action or correct the system. Note that the user must be aware that the action is taking place, which means that the level of detail (d) should be higher than level d_0 (or in other words: $d > d_0$).

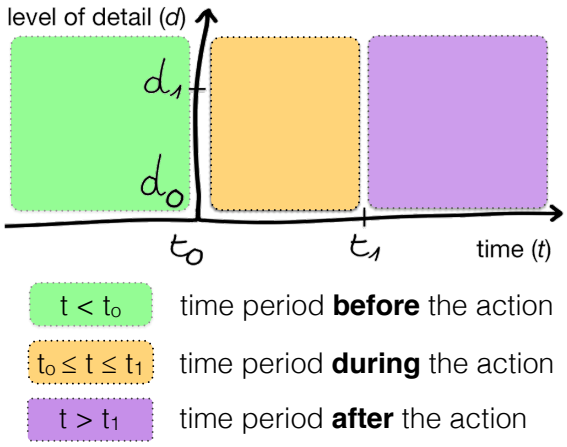


Figure 6.4: The three regions in the design space: before, during, and after the action.

6.2.2 *Relation to the Design Space for Intelligibility and Control*

This design space is a specialized subset of the overall design space for intelligibility and control that was introduced in Chapter 3 (Figure 3.3 on page 36). As shown in Figure 6.5, it provides a more fine-grained exploration of the timing dimension in

the original design space combined with a second dimension that conveys the level of detail at which information is provided. The combination of these two dimensions provides opportunities for exploring the relations between moments in time when actions begin or end, and when and how information about those actions is presented. The design space allows us to more clearly conceptualize slow-motion feedback as a specific strategy that slows down ongoing actions in order to increase the time period during which information about those actions is available.

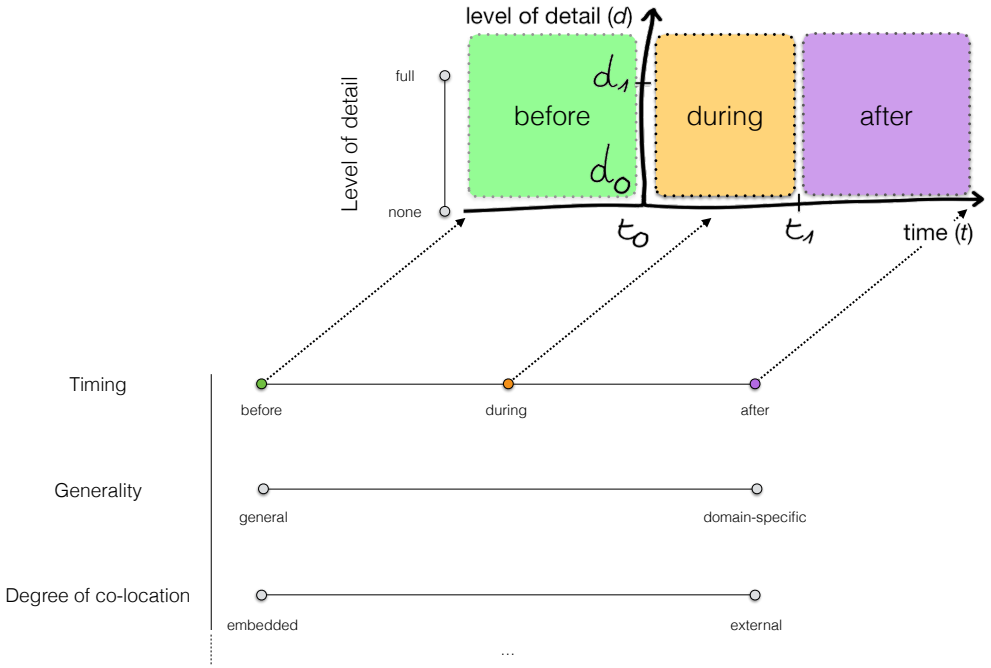


Figure 6.5: The design space combining time and level of detail (Figure 6.3) allows for a more fine-grained exploration of the timing dimension in the overall design space for intelligibility and control (Figure 3.3).

Ubicomp designers, researchers and developers can combine this design space with the overall design space presented in Chapter 3 to explore design decisions with respect to intelligibility and control. In particular, this design space can be used for fine-grained exploration of design decisions related to the timing dimension. It can be used to investigate specific moments in time, relating when actions are being executed to when and with what amount of detail information is being provided. Like the overall design space, it can be used both to analyse existing designs (as illustrated in Section 6.4) and to generate alternatives. The remainder of Section 6.2 uses this design space to analyse and differentiate between existing strategies such as feedback, intermediate feedback and feedforward and to define slow-motion feedback.

6.2.3 *Strategies Covered by the Design Space*

We illustrate the different regions in the design space (Figure 6.4) with a number of common strategies to provide information about the result of an action.

6.2.3.1 *After the Action: Only Feedback*

In graphical user interfaces (GUIs), information about the result of an action is typically only provided after the action has been completed, or in other words when $t > t_1$, as shown in Figure 6.6a. In this specific situation, information is provided in full detail ($d = d_1$). Even though the action might take some time to complete (i.e., $t_1 - t_0 > 0$), users can only intervene when they have information about what is happening. If no information is provided before the action has been completed ($d = d_0$ for $t \in [t_0, t_1]$), users have no way to prevent the action from occurring. A common way to address that problem is to allow users to revert back to a previous state, e.g., via an *undo* command. This strategy is appropriate when users explicitly trigger actions. However, it would be less desirable for systems that act autonomously and/or rely on implicit interaction, in which users can become frustrated about repeatedly attempting to revert unwanted behaviour.

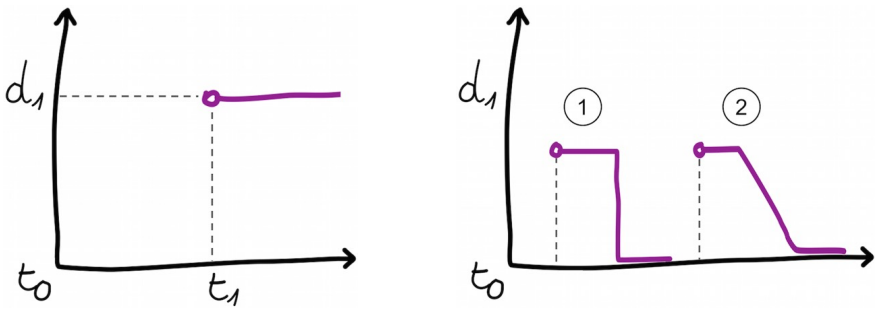
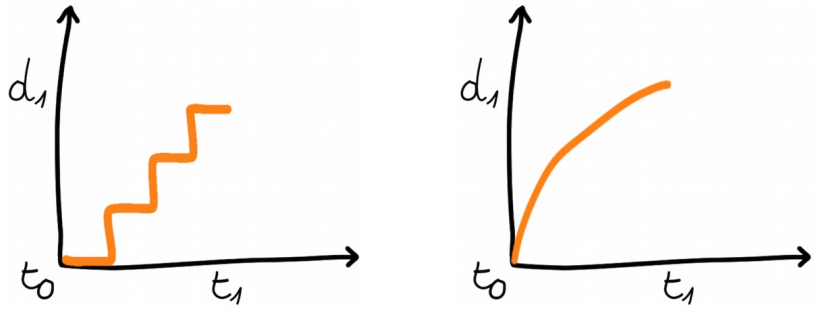


Figure 6.6: Feedback (a) and different options for the duration of feedback (b).

There are different options regarding the duration of feedback, as shown in Figure 6.6b. Feedback can be an inherent part of the user's ongoing task, and will then remain visible over time (e.g., text that has been added to a document), which is the case in Figure 6.6a. Other kinds of feedback might be temporary and disappear quite quickly, such as subtle notifications when a word processor has auto-corrected a word. Note also that in the case of Figure 6.6a, the provided information is not complete ($d_0 \leqslant d \leqslant d_1$), but can be sufficient for the user. Not all notifications are temporary though, some might be important enough to remain visible until the user deals with them (e.g., notifications about software security updates).

6.2.3.2 *During the Action: Intermediate Feedback*

Another common pattern is showing information during the execution of the action ($t_0 \leqslant t \leqslant t_1$). This allows users to keep an eye on what is happening and to intervene if necessary. This type of feedback is often used for long-running tasks to inform users about the current state of the system. While several curve shapes are possible, the level of detail is usually increased incrementally, as shown in Figure 6.7a.



(a) Incremental (b) Continuous

Figure 6.7: Incremental and continuous intermediate feedback.

Typical examples of incremental feedback are applications that allow users to preview results while they are being processed. When loading a website over a somewhat slower connection, for example, users can already see partial content coming in before the website is fully loaded. Should the user suddenly realize that this was not the website they were looking for, they can easily go back without having to wait for the entire page to be loaded. As another example, some image editing applications show incremental feedback while applying filters to an image. An example of continuous intermediate feedback (Figure 6.7b) is the OctoPocus gesture guide (Bau and Mackay, 2008). When users perform a gesture, the gesture guide continually shows how the user's input is interpreted by the system by updating the possible remaining gesture paths.

6.2.3.3 *Before the Action: Feedforward*

Information about the result of an action can also be provided before the action has been started (when $t < t_0$), as shown in Figure 6.8. In Chapter 5, we called information about the result of an action that is presented *before* the action *feedforward*. Feedforward can also remain visible or disappear once the action is being executed.

An example of this is indicating that the camera flash will go off before taking a picture, as shown in Figure 5.1 on page 73. It is also common to allow users to *preview* a change before it takes effect. In Microsoft Word, for instance, when changing the font colour of a text selection, the result is already visible on the selected text when the user hovers over the different colour buttons. Although one

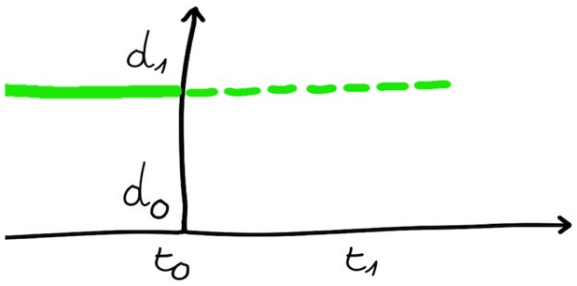


Figure 6.8: Information about the result of an action can be shown before t_0, in which case it is feedforward (see Chapter 5).

can argue over when an action actually starts in this case (when the user hovers over a button or when she clicks it?), the user still has to confirm the action before it will be executed by the system. We consider any information that is provided before executing the action, to be on the left side of the time axis ($t < t_0$).

6.2.4 *Defining Slow-Motion Feedback*

We now define slow-motion feedback using the design space. Slow-motion feedback is a *particular strategy for intermediate feedback*, where the system's actions are deliberately *slowed down* to increase awareness of what is going on and to provide opportunities for user intervention. Slow-motion feedback is less relevant for long running tasks where users have no difficulty noticing that something is happening and have sufficient time to intervene.

Slow-motion feedback is essentially about amplifying the time difference between t_1 and t_0 ($t_1 - t_0$), or in other words, the duration of an action in the user's time frame. Execution of the action is postponed by delaying t_1 to t_2 (with $t_2 > t_1$). The available time to notice that the action is happening thus increases to ($t_2 - t_0$), as shown in Figure 6.9. Designers can rely on animations (Chang and Ungar, 1993) to transition between t_0 and t_2, such as *slow-in/slow-out* in which the animation's speed is decreased at the beginning and at the end of the motion trajectory to improve tracking and motion predictability (Dragicevic et al., 2011).

It is important to note that slow-motion feedback also depends on the provision of information about what is happening. Consider the situation where there is no information provided at time t_0 ($d = d_0$). Suppose that information is only provided after a certain time t_x. In other words, there is a point (t_x, d_y) in the design space where $t_0 \leqslant t_x \leqslant t_1$ and $d_y > d_0$ (i.e., there is information available). In this case, the available time to notice that the action is happening increases from $t_1 - t_x$ to $t_2 - t_x$. A general definition of slow-motion feedback would thus be as follows:

> *Slow-motion feedback delays the system action from time* t_1 *to* t_2*. By doing so, it increases the time to notice what is going on and intervene from* $t_1 - t_x$ *to* $t_2 - t_x$ *for* (t_x, d_y) *where* $t_0 \leqslant t_x \leqslant t_1$ *and* $d_y > d_0$.

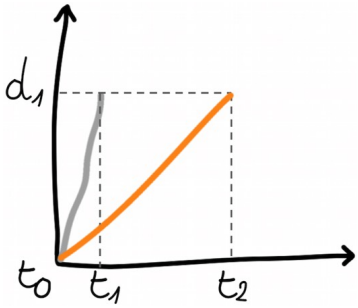


Figure 6.9: Slow-motion feedback amplifies the time to intervene by showing feedback until t_2 instead of t_1.

In the next section, we delve deeper into the example shown in Figure 6.1. We discuss how slow-motion feedback can be used to make users aware of what is happening in context-aware environments.

6.3 AN APPLICATION OF SLOW-MOTION FEEDBACK: THE VISIBLE COMPUTER

6.3.1 *Introduction*

As mentioned in Section 2.1, a key aspect of the vision of ubiquitous computing is moving computers into the background, making them *invisible* to end-users. This design ambition is present in Mark Weiser's vision (Weiser, 1991), and also can be found in related research efforts such as the EU-funded Disappearing Computer Initiative (Streitz et al., 2007). Weiser (1994) described the idea of invisible computing as follows:

> For thirty years most interface design, and most computer design, has been headed down the path of the 'dramatic' machine. Its highest ideal is to make a computer so exciting, so wonderful, so interesting, that we never want to be without it. A less-travelled path I call the *invisible*; its highest ideal is to make a computer so embedded, so fitting, so natural, that we use it without even thinking about it.

If computers are to be so natural that they become invisible in use, they will frequently need to function on the periphery of human awareness and react on implicit input (see Section 2.2.1). However, we argued that there are inherent problems in having systems act *without user intervention*, as it is difficult for computers to accurately model and respond to our behaviour. Therefore, at certain times, the system will have to involve their users, make them aware of what it is doing and allow them to intervene. In other words, the system will have to make itself *visible* to the user.

Following this argument, we explore the inverse: what if we turn the idea of the invisible computer around (Vermeulen et al., 2009a)? What would a *visible computer*,

that presents itself to the user and reveals in full detail how it works, look like? As shown in Figure 6.10, to this end, we use projectors to overlay the environment with a visualization of the different sensors and input or output devices in the environment. We highlight generated events, and show animations revealing the flow of events between sensors and devices, to illustrate what caused the system to act and what effects that system action will have on the other devices in the environment. By doing so, this technique reveals the underlying behaviour model of the environment to the user. Note that we assume that the behaviour is driven by a set of context rules. Additionally, we employ slow-motion feedback to make users aware of actions that are happening. Actions in progress can be cancelled using a simple voice command. We explored the usefulness of these visualizations in an informal first-use study (Section 6.3.5).

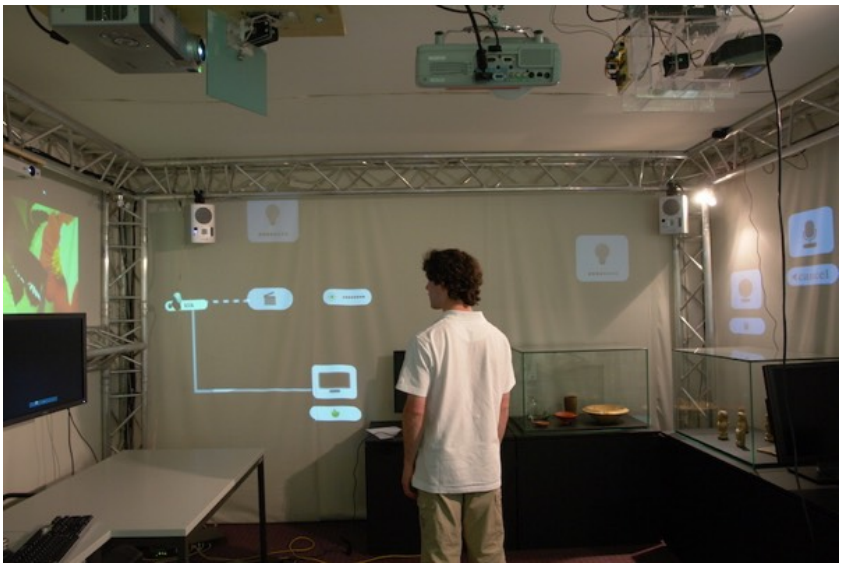


Figure 6.10: A projected visual representation of the context-aware environment shows the different devices and sensors that are present and how events trigger system actions.

Of course, constant visualizations might be distracting and contrary to Weiser's idea of calm computing (Weiser, 1991). We therefore believe that these visualizations are mainly useful as a *debug mode* for end-users. The visualizations could be hidden when users have gained more experience with the system, and called upon again whenever users have difficulties understanding the system's behaviour. However, it can be useful to always show a simplified version of the slow-motion feedback animations whenever actions are triggered.

In what follows, we provide a brief overview of related work, explain our visual behaviour representation, and present the observations from the first-use study.

6.3.2 *Related Work*

Since we overlay the physical room with additional virtual information, our system can be seen as an application of *augmented reality* (Azuma, 1997). Researchers have previously explored to use of projectors to realize augmented reality (AR) and a major advantage of this approach is that it does not require additional hardware such as head-mounted displays. For example, IBM's "Everywhere Displays Projector" is a steerable device that can project ubiquitous graphical interfaces on different surfaces in the environment (Pinhanez, 2001). Others have combined several static projectors that are stitched together in order to create a larger projection surface (Raskar et al., 2003; Chen et al., 2002; Raskar et al., 1998). Mobile projector-based systems allow the user to freely move around and control where the projector is displaying information (e.g., Raskar et al., 2004; Cao et al., 2007; Willis et al., 2011), similar to the Feedforward Torch technique presented in Chapter 5. Researchers have also investigated how these systems can support interaction by recognizing the geometry of the room and the people and objects in it (e.g., Molyneaux et al., 2007, 2012; Wilson et al., 2012).

Due to the emphasis on invisibility in Weiser's vision, ubiquitous computing systems tend to have very little support for traditional user interface concerns such as feedback, control, and indeed 'visibility' (Bellotti et al., 2002), as discussed in Section 2.2.3. Increasing awareness of these issues and the suitability of augmented reality for in-place guidance and instructions (e.g., Feiner et al., 1993), gave rise to a number of AR techniques that try to address these issues. White et al. (2007) overlay tangible objects with visual hints to allow users to quickly see what gestures can be performed with an object. Tan et al. (2001) describe a tangible AR interface in which users can get in-place help by placing special help cards next to other data cards. Sandor et al. (2005) describe a graphical language similar to ours—with lines connecting input devices to real or virtual objects that these devices manipulate—to facilitate the configuration of mixed-reality environments. Rehman et al. (2005) describe how a location-aware Ubicomp application was enhanced with augmented reality visualizations to provide users with real-time feedback. An initial user study compared Rehman et al.'s augmented version of the application with the original one. Results suggested that the visual feedback made for a more pleasant user experience, and allows users to form a better mental model, which is in line with our findings (we will come back to this later in Section 6.3.5).

Earlier work investigated how to visualize context rules for end-users. For example, iCAP (Dey et al., 2006), a design tool for end-user prototyping of context-aware applications, also represented context-aware behaviour rules in a visual manner. Similarly, Welbourne et al. (2010) specified complex location events in a visual way, using a storyboard metaphor. With our technique, however, users can see a visualization of the system's behaviour in real-time and in-situ—when and where the events take place. Although targeted towards developers, Marquardt et al. (2010) presented an interactive event visualization that showed the location and status of different sensors and devices, and events flowing between these distributed compo-

nents. Moreover, Marquardt et al. (2010) also allowed developers to slow down the visualization speed in their system to investigate device events in more detail.

6.3.3 *A Visual Representation of Behaviour*

Our technique was developed to improve understanding of context-aware features in a smart museum environment. We developed a simple visual language to provide an overview of the state of the environment at a glance. It represents relationships between sensors or devices and the actions executed by the system. When an action is executed by the system, an animation is shown to reveal links between this action and the different devices or sensors in the environment.

6.3.3.1 *Visualizing the Environment and Its Behaviour*

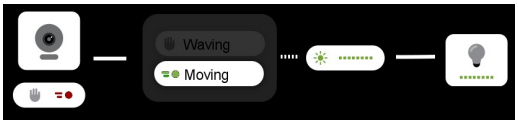
Each sensor or input/output device (e.g., a camera, speaker or display) is visualized at its physical location in the environment with an icon and a label. These icons allow users to get a view of the devices that are present in their environment. Below the icon of each *input* device or sensor, a separate label shows the possibilities of the device using smaller icons, with its current state highlighted. On the other hand, *output* devices have no separate label, they only display an icon that embeds their current state. For example, Figure 6.11a shows an icon and label for a webcam (an input device) on the left and an icon for a light (an output device) on the right. In this example, the webcam can detect a 'waving' gesture and motion in general, as indicated by the small icons in the label. The motion detection state is currently active and therefore highlighted. The light's state corresponds to its current intensity and is displayed under the light as a horizontal bar.

We define a *trajectory* as a visualization between two or more objects in the environment. Trajectories visualize connections between different devices and system actions, and consist of five parts:

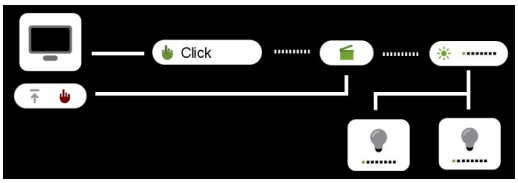
1. a source device;
2. the event that occurred at this device;
3. (a possible condition necessary to trigger the action);
4. an action to be executed;
5. one or more target devices that are impacted by the action.

Executing one action can also result in an event that triggers another action, in which case we would show a sequence of multiple actions in step 4 (see Figure 6.11b). Between each of these parts, lines are drawn. Dotted lines are used between events and actions, while connections between devices and other objects use solid lines.

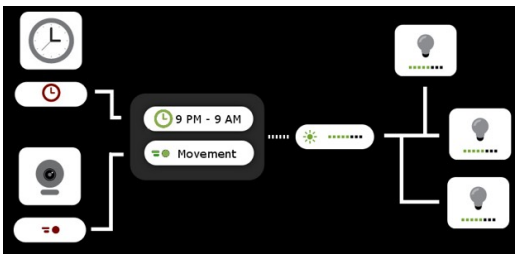
Consider the trajectory shown in Figure 6.11a. Here the webcam detects motion, which triggers an action that turns on the lights. This action, in turn, impacts the light on the right side of the figure. Note that the small state icons are shown again, together with a textual description. The 'Waving' state is shown semi-transparently



(a) Motion in front of the webcam (input) triggers the light (output). The event 'waving' of the webcam is now inactive, but could trigger another action.



(b) A chain of events: touching the screen results in a movie being played (on the same screen). This, in turn, results in the lights being dimmed.



(c) Conditions: the lights are turned on when motion is detected (through a camera) *and* when it is dark outside (between 9 PM and 9 AM).

Figure 6.11: Example trajectory visualizations.

to indicate that it is not active. A bit further to the right, a graphical representation of the action is shown, connected to the light it turns on. The lines in a trajectory will be animated from source to effect, thereby possibly spanning multiple surfaces. Device icons and labels will always be shown, even if they are not active (non-active icons are displayed semi-transparently). Other labels (e.g., action labels) only become visible when the connecting line crosses them. With respect to the design space introduced in Section 6.2, the projected visualizations thus show an increasing level of detail over time. Animations will slowly fade out after they have been completed.

As mentioned before, trajectories can also visualize multiple actions which are triggered in sequence. Figure 6.11b shows a trajectory with two sequential actions. In this situation, touching the screen causes a movie to be played on this screen. The action of playing a movie will itself cause another action to be executed: one that dims the lights for a better viewing experience. It is possible to visualize more complex rules that combine multiple sensors using boolean operators (e.g., AND, OR). Figure 6.11c shows a trajectory with an *and* condition that determines when

the system will turn on the lights. This rule will only fire if it is between 9 PM and 9 AM (i.e., when it is dark outside) *and* if the camera has detected motion. To represent *and* and *or* conditions, we use the visual representation proposed by Pane and Myers (2000), which is also used in iCAP (Dey et al., 2006): elements in *and* conditions are split vertically, while elements in *or* conditions are split horizontally.

6.3.3.2 *Overriding System Actions: The Cancel Command*

Figure 6.12 shows a visualization of the cancel command in action. We decided to implement a voice-controlled cancel command, although other ways to invoke this command (with a suitable icon to represent that action) are possible. Since the cancel command is voice-controlled, it is visualized using a microphone icon. The only possible state is an invocation of the cancel command when the word "cancel" is recognized, as indicated in the corresponding label. When an action is cancelled the microphone will turn around and shoot at the icon corresponding to the effect of the action, resulting in this icon being destroyed. The shooting animation can again span different surfaces to reach its target. This kind of visual feedback shows users in a playful way that the effect that the action had on the environment has been undone. Note that explicitly visualizing the cancel command, reveals it as an action possibility to users, since discoverability and visibility of possible actions are common problems in speech interfaces (Norman, 2010).

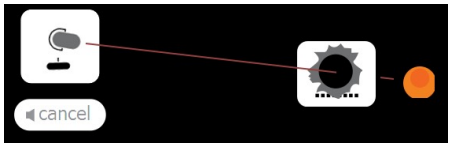


Figure 6.12: When the action 'light off' is cancelled, the microphone destroys the light icon.

6.3.3.3 *Expressiveness and Limitations*

The visual notation was deliberately kept simple. It mainly targets systems that encode their behaviour as a list of if-then rules, which is a common approach to realizing context-awareness (Dey et al., 2006). Our behaviour representation has two main shortcomings. First, we are currently unable to visualize the reasoning behind machine learning algorithms, another frequently used approach to realize context-aware systems. Secondly, as with any visual language, scalability is an issue. When the notation would be used to visualize a very complex network of connected sensors and devices, the result could become too complex to comprehend for users and could require additional filtering mechanisms.

6.3.4 *Implementation*

We use several static and steerable projectors to overlay the physical environment with our graphical representation. For details on how to set up this type of system,

we refer to the existing literature (e.g., Pinhanez, 2001; Raskar et al., 2003; Chen et al., 2002; Sukthankar et al., 2001). An overview of our architecture is provided in Figure 6.13.

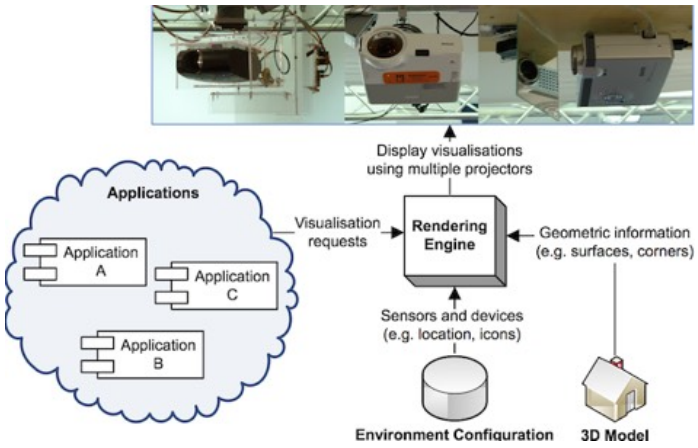


Figure 6.13: Software applications in the context-aware environment can send requests to the rendering engine to make their behaviour visible to end-users.

The most important software component in our system is the *rendering engine*. It is implemented as a central service using the Windows Presentation Foundation toolkit[1]. The engine is responsible for overlaying the environment with a visualization of all devices and sensors, and for showing animated *trajectories* between these elements when a software application executes an action. To do so, it relies on a 3D model of the environment and an environment configuration file. The 3D model of the environment is used to determine which of several steerable and static projectors need to be used and what image corrections need to be applied to display the annotations. The configuration file encodes the position of each device and sensor in the environment, together with their icons, possible states and a number of predefined trajectories. When software applications need to visualize a change of state in a device or the execution of a certain action, they send a request to the rendering engine containing an XML description of the required state change or trajectory.

Our current implementation was deliberately kept simple as we were mainly interested in exploring the potential of this technique. For a realistic implementation, the rendering engine needed to be extended in two ways. First, it should be able to receive events from an underlying context-aware system, e.g., executed rules and triggered sensor events (e.g., "motion detected"). This would be possible by integrating the rendering engine with a ubicomp framework such as PervasiveCrystal, which we describe later in Chapter 7. Secondly, the rendering engine should be able to automatically find a path between different elements based on their position in the environment, without causing too much visual clutter (e.g., crossing lines).

1 http://www.windowsclient.net/

6.3.5 *Evaluation*

6.3.5.1 *Participants and Method*

We ran an informal first-use study to investigate the suitability of our technique for understanding the behaviour of a context-aware environment. The goal of the study was to explore how in-situ visualizations could help users to understand connections between different devices and sensors in the room. The experiment was carried out in a realistic ubicomp environment: a room simulating a smart museum that featured different kinds of sensors and devices. We deployed a number of sample applications on the room's server that controlled other devices in the environment based on context changes (e.g., controlling the lights based on motion detection using a camera). Applications were developed with Processing[2] and communicated with each other and the rendering engine over the network using the Web-to-Peer (W2P) messaging protocol (Vanderhulst et al., 2007). The intensity of the lights in the environment could be changed by sending simple UDP messages to a custom-built piece of control hardware. Two of the sensors used during the experiment were implemented using the Wizard of Oz technique (Kelley, 1984) to avoid recognition problems: the voice-controlled cancel feature and the webcam motion detection sensor.

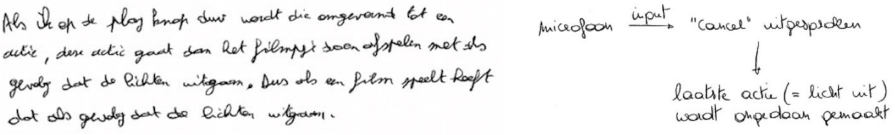
Five voluntary participants from our lab participated in the study, with ages ranging from 24 to 31 (mean = 27.8); three were male, two female. All participants had general experience with computers. Four out of five had experience in programming, while the fifth participant had no experience with programming and was a historian. Each individual study session lasted about 40 minutes. First, participants were asked to read a document that provided a brief overview of our visual language and explained when visualizations would be projected in the room. The document also explained that the goal of the projected visualizations was to help people understand what was happening in a smart environment and allow them to intervene if necessary. Afterwards, participants were presented with three situations in which they had to understand the environment's behaviour using the visualizations. All participants followed the same order of tasks. After completing the experiment, participants were interviewed and asked to fill out a questionnaire, which can be found in Appendix B.3. The three tasks participants had to perform during the study were:

- *Task 1*: Participants were asked to press a play button on a touch screen, after which a movie would start to play on one of the walls. This, in turn, triggered an action that turned off the lights to provide a better viewing experience.
- *Task 2*: Participants were given the same instructions as in the first task, but were also told to find a way to turn the lights back on afterwards. They were expected to use the *cancel* functionality to achieve this effect, which was briefly explained in the introductory document.

2 http://www.processing.org/

- *Task 3*: In the last task, participants were asked to walk up to a display case and were told that they would notice a change in the environment. The display case was equipped with a webcam for motion detection, which would turn the lights on or off depending on the user's presence.

Participants were allowed to explore the system and perform each task several times until they felt that they had a good understanding of what was happening. After completing a task, participants received a blank page on which they had to explain how they thought that the different sensors and devices were connected. This allowed us to get an idea of each participant's mental model of the system. Participants were free to use drawings or prose (or a combination of both). Two examples of explanations that participants created during the study are shown in Figure 6.14.



(a) (b)

Figure 6.14: Two examples of explanations that participants created during the study: (a) explaining why the lights go out in task 1 and (b) explaining how the cancel feature works in task 2.

6.3.5.2 *Observations and Discussion*

Participants were generally happy with our visualizations. One mentioned that he found it "convenient to follow the lines to see what is happening", while another said: "it was clear to see which action had which effect". In the post-experiment questionnaire, participants ranked our technique highly for being useful to understand and control what happens in a context-aware environment and for *not* being confusing. In general, participants indicated that they understood how to use our visualization technique, that they found the visualization easy to understand and that it provided them with the information they wanted to know.

When asked to explain how the system worked in their own words, four out of five participants (including the participant without a technical background) described the system's behaviour correctly for each of the three tasks. This could suggest that explicitly visualizing the behaviour of a context-aware environment helps users to form a correct mental model, which would be in line with the findings of Rehman et al. (2005). However, further studies with a wider variety of tasks and a larger group of participants are needed, to confirm these initial findings.

The study also revealed a few shortcomings in our current prototype. Three participants reported problems with recognizing some features of devices or sensors using their icons. Both the touch screen and cancel icons were found to be unclear.

We observed that several participants experienced difficulties in invoking the cancel feature. In addition to the unclear icon, the fact that several participants were unfamiliar with speech interaction could be another factor contributing to these problems. One participant mentioned that he felt "uneasy using a voice-controlled command", because "he was used to clicking". The one participant who did not manage to correctly describe the system's behaviour in all tasks, also failed in the second task that used the cancel feature.

Additionally, we observed that some participants experienced difficulties with keeping track of visualizations across multiple surfaces. Sometimes the visualization would start outside participants' field of view, which caused them to miss parts of the visualization. A possible solution is to use spatial audio to guide users' attention to the area of interest, which may help to guide users' attention to the area of interest (Butz and Krüger, 2006). One participant commented that she sometimes received too much information, which confused her. She referred to the first task, in which a 'click' on the touch screen was visualized as causing the movie to start playing. With respect to the *level of detail* (see Figure 6.3), it might be useful to decrease or increase the level of detail based on the type of action visualized by the system. For example, it might be useful to show less-detailed visualizations for actions which occur often and are obvious to users.

6.4 APPLICATIONS OF SLOW-MOTION FEEDBACK

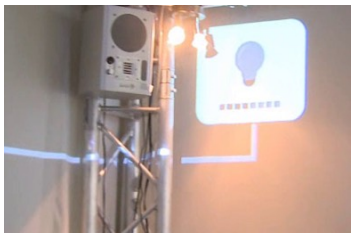
In this section, we look into different applications of slow-motion feedback, and analyse them further using the design space introduced in Section 6.2.

6.4.1 *Visualizing Behaviour and Causality: The Visible Computer*

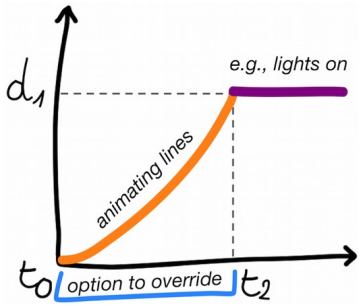
The technique presented in Section 6.3 uses slow-motion feedback to improve awareness of system actions and what causes these actions to occur. By slowing down the visualizations, users are given the option to intervene. Consider again the scenario where a motion sensor causes a light to be turned on (Figure 6.11a). The system's action (turning on the lights) is slowed down here and timed to exactly coincide with the moment in time at which the animated line reaches the light (Figure 6.15).

6.4.2 *System Demonstration*

Ju et al. (2008) have applied slow-motion feedback to address interaction challenges in implicit interaction. Their Range interactive whiteboard uses slow-motion feedback to transition between ambient display mode and whiteboard mode based on the user's proximity to the display. It shows an animation moving all content from the centre of the board to the sides when a user steps closer. This happens slowly enough so that users both notice it, and have sufficient time to react if it was not what they wanted (Figure 6.16). Users can override the automatic action of making space by grabbing content and pulling it back to the centre. When users notice the



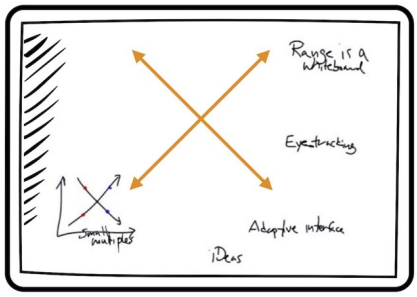
(a) The Visible Computer



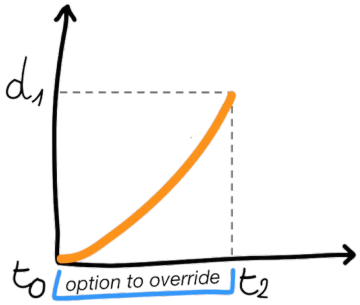
(b) Design space

Figure 6.15: Using slow-motion feedback to visualize system behaviour.

contents of the whiteboard moving to the side of the display (Figure 6.16a), they can predict what will happen and possibly override the system's action.



(a) System demonstration



(b) Design space

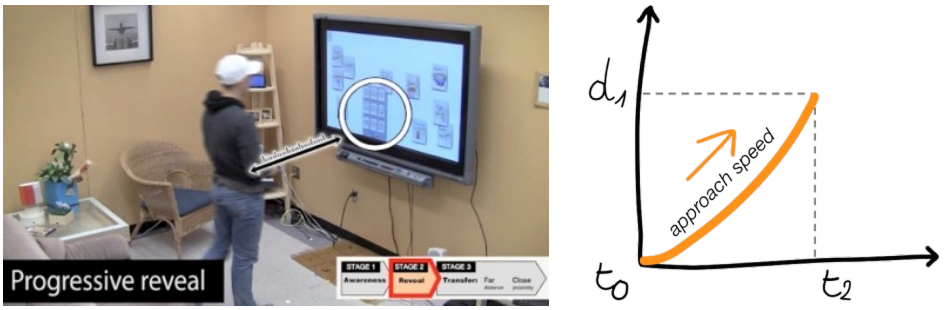
Figure 6.16: Ju et al. use slow-motion feedback in the Range whiteboard to provide users with awareness of system actions and provide the opportunity to override these actions.

6.4.3 *Progressive Feedback*

The gradual engagement design pattern for proxemic interactions (Marquardt et al., 2012a) relies on proximity to facilitate information exchange between devices. It is comprised of three stages in which more information is shown as the user's engagement with the system increases (e.g., by approaching a device). Marquardt et al. assume that users will approach or orient themselves towards a device when they are interested in interacting with it.

An interesting feature here is that users can control the speed at which information is revealed. The faster users approach a device, the faster information is shown, which realigns the system's time frame with their own (Figure 6.17). In this case, the

natural hesitation of novices and the rapid approach of experts might have exactly the intended results.



(a) Gradual Engagement (source: Marquardt et al., 2012a)

(b) Design space

Figure 6.17: Progressive feedback gives users control over the speed at which information is revealed.

6.4.4 *Postponed Feedback*

Ju et al. (2008) have applied an additional strategy for slow-motion feedback. Their electronic whiteboard performs automatic stroke clustering in the background while the user is drawing. The system provides feedback about the clusters by surrounding strokes in dotted light-gray bounding boxes. However, to avoid interrupting users while they are drawing, this feedback is only shown when the user steps back. When users notice a misclustering, they can override the system's action by redrawing the outline. The interesting aspect of this approach is that the action and feedback cycle is shifted into the future (Figure 6.18). Instead of increasing the time between start and end of the action, users are only made aware of the action when they can be interrupted (at time t_2), similar to attentive interfaces (Vertegaal, 2003).

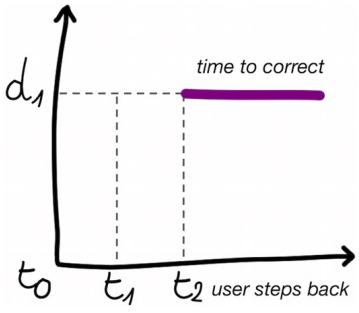
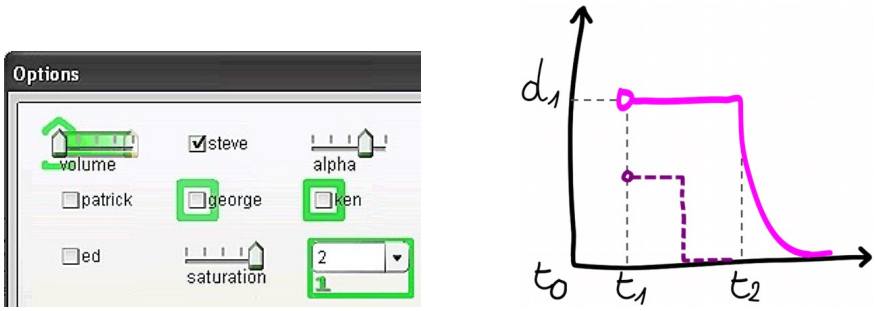


Figure 6.18: Postponed feedback is only shown after t_2, even though the action was already completed at t_1.

6.4.5 *Emphasizing Change*

Finally, Phosphor by Baudisch et al. (2006) visually augments GUI widgets to emphasize changes in the interface and leave a trail to show users (in retrospect) what just happened. Phosphor increases the already existing feedback's level of detail (an increase in d) and the time that it is shown to the user (an increase in t), as illustrated in Figure 6.19b. Note, however, that Phosphor does not postpone the action.



(a) Phosphor (source: Baudisch et al., 2006) (b) Design space

Figure 6.19: Phosphor increases both the level of detail and time.

6.5 DISCUSSION

In this chapter, we introduced slow-motion feedback, a technique to increase awareness of system actions and simultaneously provide users with opportunities for control. We defined slow-motion feedback using a simple design space, and used that design space to understand traditional feedback strategies and analyse existing applications of slow-motion feedback. In particular, we discussed our visible computer technique in which we explored a visual representation of the behaviour of a context-aware environment. An informal first-use study suggested that these visualizations might indeed improve understanding and support users in forming a reliable mental model.

There are also a number of implications of applying slow-motion feedback. An open issue is how slow-motion feedback can be applied to time-critical tasks, as it might have a negative effect on the overall task completion time. While this will be negligible in most cases, when applied to several sequential micro-interactions, the cumulative effect over time might be too large to ignore. In addition, more work is needed to take into account diverse groups of users. If the speed of slow-motion feedback is fixed for all users, there will be situations in which the provisioning of feedback will be either too slow (e.g., for experts) or too fast (e.g., for novice users). We see the biggest potential in approaches that allow the user to control the speed at which information is provided.

Having covered techniques for intelligibility and control *before* actions (Chapter 5) and *during* actions (this Chapter), we now turn our attention to techniques to provide intelligibility and control *after* an action has occurred. In Chapter 7, we further explore the *Timing* dimension introduced in Chapter 3 with our work on providing automatically generated answers to 'why?' and 'why not?' questions (Vermeulen et al., 2010).

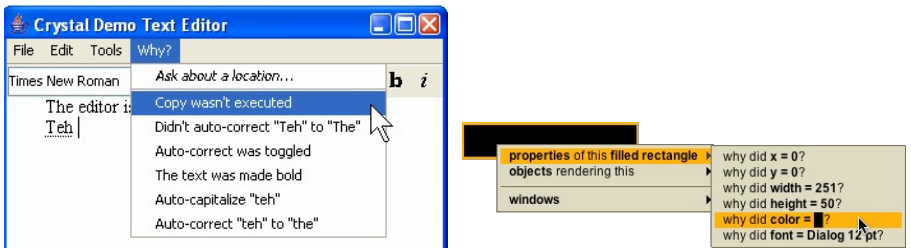
7

ANSWERING WHY AND WHY NOT QUESTIONS ABOUT CONTEXT-AWARE APPLICATIONS

7.1 INTRODUCTION

7.1.1 *Answering Why and Why Not Questions to Improve Understanding*

According to Bellotti and Edwards's original definition, intelligible context-aware systems should be able to represent to their users "what they know, how they know it, and what they are doing about it". One approach to supporting intelligibility is to provide textual explanations that inform users of what the system did, or why it acted in a certain way (see Section 3.2.1). The use of explanations originated in research on making intelligent systems understandable for end-users (e.g., Gregor and Benbasat, 1999). In particular, explanations answering 'why?' and 'why not?' questions have been found to be useful for dealing with complexity in a range of scenarios, including helping users understand complex end-user applications (Myers et al., 2006) and helping both end-user programmers and professional programmers debug their programs (Ko and Myers, 2004, 2009). Indeed, Norman notes that "people are innately disposed to look for causes of events, to form explanations and stories" (Norman, 2013b, pg. 59). By allowing users to pose 'why?' and 'why not?' questions, we can improve understanding by taking advantage of people's natural predisposition to find causes for events.



(a) Crystal (Myers et al., 2006) (b) The Whyline (Ko and Myers, 2009)

Figure 7.1: Two examples of the use of why questions to improve understanding: (a) in complex end-user GUI applications, and (b) in debugging complex applications (source: images extracted from the respective papers).

Figure 7.1a shows how Myers et al. (2006) provide answers to 'why?' and 'why not?' questions in a word processor to help end-users understand complex behaviours and interdependencies among various of the application's features. Users can go into the *why menu* and select a question of interest, after which they will receive an automatically generated answer to that question. For example, the user

could ask why 'Teh' wasn't auto-corrected to 'The'. Whenever users are confused, they can pose questions about recent events that occurred, or can ask location-specific questions (e.g., "Why is this text bold?"). As another example, Figure 7.1b shows The Whyline (Ko and Myers, 2009), which allows programmers to directly ask why questions about the program's behaviour and view answers to those questions in terms of relevant runtime data. In developing this technique, Ko and Myers were motivated by the tendency of programmers to naturally ask 'why?' and 'why not?' questions during debugging. Developers can ask questions about runtime objects, such as what caused a visual object to have a specific background colour. The Whyline then points the developer to the line of code responsible for that behaviour.

7.1.2 *Scope and Chapter Outline*

In this chapter, we describe a framework to support 'why?' and 'why not?' questions in context-aware systems. A number of studies (Lim et al., 2009; Lim and Dey, 2009) have reported that supporting these questions in context-aware systems would result in better understanding and stronger feelings of trust. By asking 'why?' and 'why not?' questions, arising respectively from *unexpected* events that *occurred* or *expected* events that *did not occur*, users can gain a better understanding of the internal working of the system. However, supporting 'why?' and 'why not?' questions is not trivial for complex context-aware systems with several distributed components. Existing desktop implementations such as Crystal (Myers et al., 2006) cannot be easily integrated into ubicomp frameworks, since the assumptions underlying these implementations—such as having a single machine from which events originate—rarely hold true in ubicomp environments. We explain here how we have extended the existing ubicomp framework ReWiRe (Vanderhulst et al., 2008) with support for answering why and why not questions. This resulted in *PervasiveCrystal*, a framework to develop distributed context-aware applications that support asking and answering 'why?' and 'why not?' questions.

Note that why questions are mostly useful for past actions. Regarding the *timing* dimension that was introduced in Section 3.3, we can thus classify this technique as providing intelligibility and control *after* the action. Why questions allow the user to receive more detailed information about what caused that action to occur, and can help users in forming a correct mental model of the system. When combined with control mechanisms, users are not only able to understand the system's reasoning but also revert to a previous state if a certain action was undesired.

We start by illustrating how PervasiveCrystal works by means of a short usage scenario. We then explain the basics of the ReWiRe framework and how its easy-to-query behaviour model allows us to trace events across distributed components. Next, we describe related work. We continue by explaining how we use the behaviour model to generate both why questions and answers to these questions and provide users with different control mechanisms. Finally, we describe an exploratory user study and discuss limitations of our approach and possible extensions.

7.2 USAGE SCENARIO

In this walkthrough, we will follow Bob, a visitor to a context-aware smart museum environment built using PervasiveCrystal. As Bob enters, he receives a mobile museum guide that provides information about artefacts in the museum, and can also be used to interrogate and control the environment. We explain with a few simple examples how Bob can make use of 'why?' and 'why not?' questions.

7.2.1 *Posing Why Questions*

Upon entering the museum, Bob was told that the museum features interactive displays that react to motion. When Bob approaches one of these displays during his visit, he waves in front of the screen to play a movie, as shown in Figure 7.2 (*scene 1*). However, when he does that, the lights also go out. Bob does not understand why this happens and is confused (*scene 2*). The museum relies on a rule-based system to react to context changes (*scene 3*). One of these rules is configured to play a movie when motion is detected by a camera. However, there is also another rule that turns off the lights whenever a movie is playing to provide users with a better viewing experience. After the first rule was executed, its effect (playing a movie) caused the second rule to execute as well and as a result, turn off the lights.

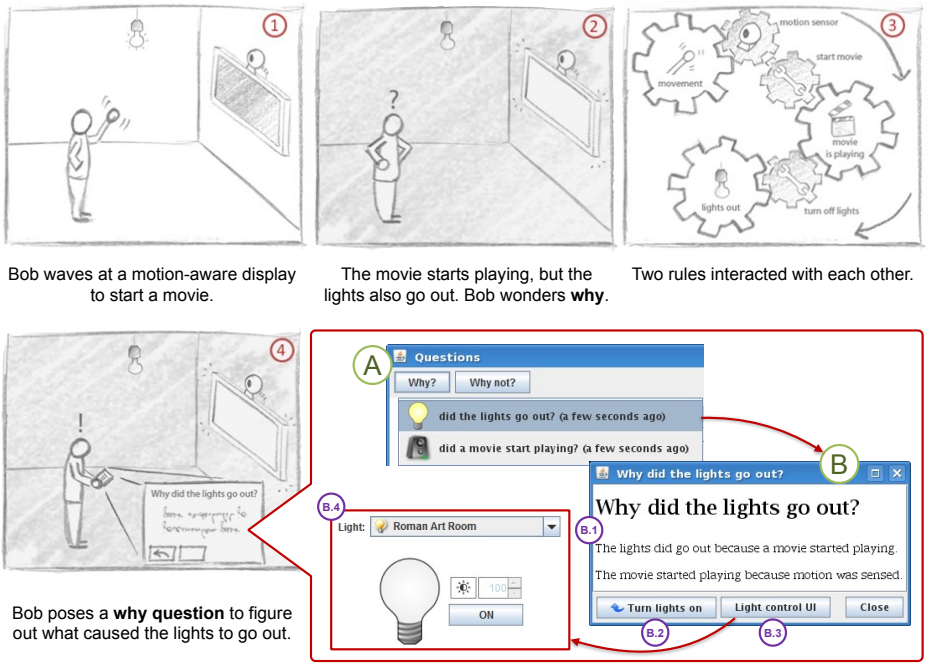


Figure 7.2: Posing a *why* question: PervasiveCrystal shows available questions, based on events that recently took place in the environment (*A*). Answers are generated by linking events to what caused them to happen (*B.1*). Additionally, users have two means for correcting the environment's behaviour: they can *undo* operations (*B.2*) or invoke fine-grained control user interfaces (*B.3*), in this case: a light control user interface (*B.4*).

Bob remembers he can use the *why menu* to ask questions about the smart museum's behaviour. As seen in Figure 7.2 (*A*), the why menu shows a list of available questions about events together with a representative icon. PervasiveCrystal automatically generates the list of questions by tracking events that occurred (e.g., lights that are switched off). The questions are presented in reverse chronological order (questions about the most recent events come first). Bob then selects the question "Why did the lights go out?", and receives an answer that briefly explains what caused both rules to fire (*B*). PervasiveCrystal can generate these answers by linking events to what caused them to happen. In this case, the system knows that the lights went out because a movie started playing. When sufficient information is available, the system will explain the entire execution trace of the event (*B.1*). Here, the system also includes an explanation of why the movie started playing in its answer: "because motion was sensed". Explanations for chains of interacting rules can be of arbitrary length.

Besides helping Bob to understand why the system has taken a certain action, PervasiveCrystal also allows Bob to intervene and correct unwanted behaviour. Within answer dialogues, such as the one in Figure 7.2 (*B*), users have two ways of controlling the system. First, the left button allows users to *undo* unwanted actions (*B.2*). In Bob's case, clicking the button will turn the lights back on, thereby undoing the action taken by the system. Secondly, PervasiveCrystal provides users with more fine-grained control user interfaces to correct undesired behaviour. The second button from the left (*B.3*), allows users to invoke a task-specific control user interface. Here, Bob can bring up the light control user interface, providing him with more options such as the specific intensity of each of the lights in the museum (*B.4*). PervasiveCrystal achieves this by annotating events with related user tasks (e.g., controlling lights, playing media) and their respective user interfaces.

7.2.2 *Why Not Questions*

Later, Bob returns to the display and tries to start the movie again, as shown in Figure 7.3 (*scene 1*). However, this time, nothing happens. Bob does not understand why the system acts differently now (*scene 2*). Behind the scenes, the camera motion sensor never reported that it detected motion and as a result, the rule that is responsible for playing the movie never got executed (*scene 3*). Bob then remembers that he can also pose *why not* questions. For this, Bob uses the *why not menu*, as shown in Figure 7.2 (*A*), which works in a similar way as the why menu that we discussed before. Instead of listing questions about events that did occur, however, the why not menu presents users with a list of questions about expected events that *did not occur*.

When expected events do not take place, the cause is often an unexecuted rule which was supposed to trigger the event. PervasiveCrystal keeps track of which rules can trigger which events, and analyses unexecuted rules to fill the why not menu with a list of questions about events could have taken place (but did not). Based on the available information about a rule, it then tries to determine why these rules did not execute.

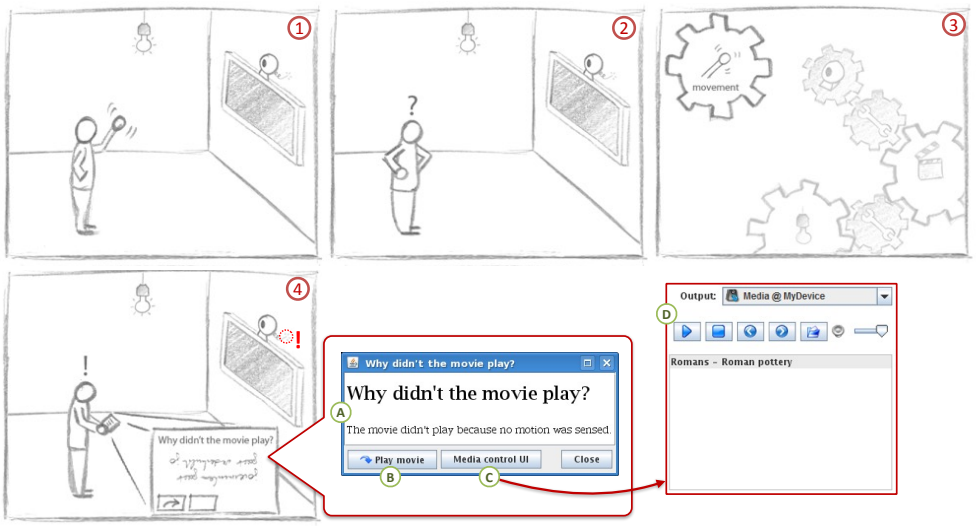


Figure 7.3: Posing a *why not* question: This time, nothing happens when Bob moves in front of the display. By asking a why not question, Bob is able to figure out that the system did not sense motion (*A*). He then notices that the camera cable is unplugged. Bob is again provided with different ways to control the environment. He can use the *do* command to force the system to play the movie anyway (*B*), or bring up the media control user interface (*C-D*).

Bob proceeds by selecting the appropriate question in the why not menu: "Why didn't the movie play?" (*scene 4*). The system responds by saying that no motion was sensed, as shown in Figure 7.3 (*A*). Bob then figures that something must have gone wrong with the motion detection. He notices the camera, and upon closer investigation, figures out that the camera cable has been unplugged (*scene 4*).

The 'why not?' answer dialogues again provide users with two means for controlling the system behaviour. Just like with why questions, users can invoke a fine-grained control user interface (*C-D*). The undo command is replaced by a *do* command that allows users to force the system to execute an action (*B*). *Undo* is available for *why* questions, while *do* is available for *why not* questions. Bob decides that he wants to see the movie anyway, and forces the system to *do* the action by clicking the "Play movie" button (*B*).

7.3 RELATED WORK

In this section, we provide an overview of related work. We mainly focus on the use of explanations (including why questions) in context-aware systems, and other frameworks to support why questions.

As mentioned before, the use of text-based explanations originated in research on making systems that employed artificial intelligence (AI) techniques understandable for end-users, such as expert systems (Gregor and Benbasat, 1999) or recommender

systems (e.g., Cramer et al., 2008). Given that context-aware systems employ similar adaptive and complex behaviour, researchers started applying explanations to context-aware systems as well. One of the first explorations into the use of explanations for context-aware systems was the 'Intelligent Office System' (IOS) (Cheverst et al., 2005). IOS learns the user's behaviour in an office environment using an underlying fuzzy decision-tree algorithm to support appropriate proactive behaviour. Cheverst et al. investigated a number of ways to improve comprehensibility, scrutability and control. In particular, whenever a rule is triggered, the system pops up a dialogue showing its confidence level for that rule together with a 'why' button that allows users to receive an explanation for the system's behaviour. The explanations provided by IOS are of the form "if Temp = cold then Fan -> off (confidence level: High)". Users can also consult a context history which lead the system to learning that rule. Compared to our explanations, the ones IOS provides are more technical and detailed. Although more details might certainly be useful, we observed in a first version of PervasiveCrystal that users had difficulties grasping and differentiating between rules, events, conditions and actions.

Similarly, the PersonisAD framework (Assad et al., 2007) provides *scrutable* user models that are able to provide explanations to help users understand and control how their personal information is used in ubicomp environments. García Frey et al. (2012) explored the integration of model-driven-engineering with explanations for context-aware, adaptive user interfaces. Explanations are used to provide adaptive help and assist users with issues such as where to find specific functionality. They also provide why questions, but mostly use those to explain design decisions, e.g., "why is the list of engine types sorted by price" (García Frey et al., 2013). Panoramic (Welbourne et al., 2010) is a design tool that enables end-users to specify and verify complex location events for location-aware applications. Panoramic can also explain *why* a certain location event did not occur, and additionally provides traces of historical sensor data.

Tullio et al. (2007) studied explanations for inferences of a person's interruptibility on door displays. The door displays showed the different sensors contributing to the interruptibility estimates of the person in that office. Tullio et al. found that lay users were able to understand in general how the door displays worked, although some people had preconceived notions about the system's behaviour that were difficult to change (Tullio et al., 2007). Lim et al. (2009) investigated if why questions could be used to improve understanding of context-aware systems. Their results suggest that allowing users to pose 'why?' and 'why not?' questions about the behaviour of a context-aware system would result in better understanding and stronger feelings of trust. In the study, a comparison was made between different types of questions users could ask about a context-aware system. Lim et al. found 'why?' and 'why not?' questions to be the most effective, as opposed to 'what if' and 'how to' questions that did not contribute much to users' understanding of the system or their perception of trust. Nevertheless, based on our own observations, we feel that why questions should be combined with information that is provided before and during system actions (see Chapters 5 and 6) to allow users to anticipate what the system will do. This was also mentioned by one of the participants in our

study (Section 7.6). In a later paper, Lim and Dey (2009) investigated the different information demands users have for context-aware applications under various situations. They recommend that 'why?' questions should be made available for all context-aware applications, while 'why not?' questions are more useful for specific contexts such as goal-supportive or high risk tasks.

After we developed PervasiveCrystal, Lim and Dey (2010) introduced the intelligibility toolkit, an extension to the Context Toolkit (Dey et al., 2001). Their toolkit provides automatic generation of explanations for eight different types of questions, including 'why?' and 'why not?' questions. In addition to rule-based context-aware applications, the toolkit supports different machine learning algorithms, such as decision trees, naïve Bayes and Hidden Markov Models. An interesting feature is the use of *reducer* components that can simplify explanations for end-users. With PervasiveCrystal, we kept explanations simple by design, but Lim and Dey's toolkit provides a more flexible architecture.

Finally, researchers have also explored the use of 'why?' and 'why not?' explanations to aid end-users in understanding and modifying machine-learned systems (Kulesza et al., 2009, 2013). For example, Kulesza et al. (2009) used 'why?' questions in an email client with an automatic filing feature, which learned from the user's past email filing behaviour. They provided both text-based and visual explanations that showed the weight of each word in the email contributing to the filing action. What is especially interesting here is the use of explanations to aid in debugging and modifying machine-learned programs. The insights obtained in this research could also prove useful in employing explanations to help end-users modify and personalize context-aware applications.

The next two sections provide details on PervasiveCrystal's architecture and implementation. First, we describe PervasiveCrystal's annotated behaviour model, after which we explain how that model is analysed at runtime to generate questions and answers.

7.4 THE BEHAVIOUR MODEL

PervasiveCrystal relies on a behaviour model that captures and defines how different applications react to context changes. This behaviour model is the core component that makes posing and answering 'why?' and 'why not?' questions possible. PervasiveCrystal was not built from scratch, but was developed as an extension to the existing ReWiRe ubicomp framework (Vanderhulst et al., 2008). After providing a short introduction to ReWiRe, we describe how we modified and annotated ReWiRe's behaviour model to support why questions.

7.4.1 *ReWiRe: A Framework for Rewiring Context-Aware Ubicomp Applications*

ReWiRe is a framework for easily deploying multiple context-aware applications in a smart environment, with user interfaces, applications and services that can be distributed over multiple devices at runtime. An example of an application built

with ReWiRe is a painting application on an interactive whiteboard, where users can select different painting tools in a tool palette on their mobile device (Vanderhulst, 2010).

Figure 7.4 shows one of the key components that ReWiRe uses to support this kind of flexibility: a distributed environment model that represents the current state of the environment and can be queried by all applications. The environment model is an instance of the environment ontology, and is created and updated at runtime as well as linked to software components and data objects. Changes at runtime that influence the environment model include users or devices that join or leave, applications that are added or removed, and context or location updates. ReWiRe relies on OSGi (OSGi Alliance, 2003) to provide a modular execution and deployment environment for different context-aware applications and services.

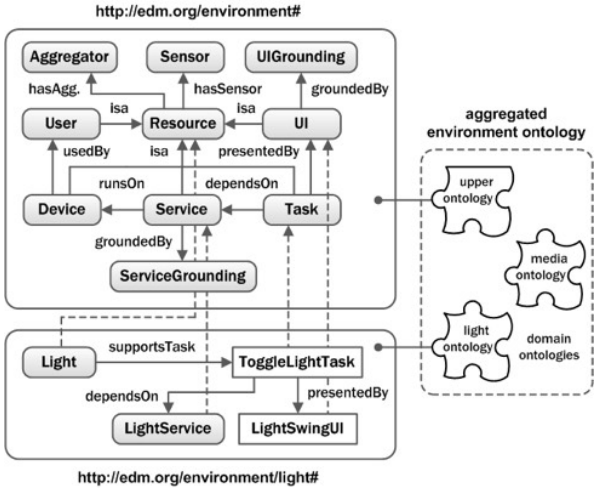
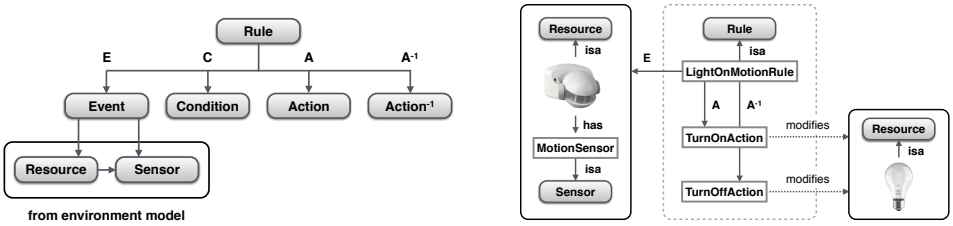


Figure 7.4: ReWiRe's environment ontology, of which an instance is created dynamically at runtime to represent the context-aware environment (image source: Vanderhulst, 2010).

To react to context changes, ReWiRe relies on a dynamic behaviour model that defines a set of context rules. Just like the environment model, ReWiRe's behaviour model is an instance of an ontology (Figure 7.5a). A behaviour rule is specified in terms of an *event*, an optional *condition*, and an *action*. The action of an Event-Condition-Action (ECA) rule (Act-Net Consortium, 1996) is triggered whenever their event fires and their condition holds true. Additionally, ECA rules can have an optional inverse action (ECAA^{-1}). Rules with inverse actions can be *undone*, and thus make it possible to return to a former state. This is achieved by caching the execution context of a rule's action (i.e. environment properties relevant for the rule) and passing this context as input to a rule's inverse action. Figure 7.5b shows an instance of a specific behaviour rule.



(a) ReWiRe's behaviour model

(b) A rule that connects a motion sensor to a light

Figure 7.5: ReWiRe's behaviour model: (a) the behaviour model consists of ECAA^{-1} rules where an event is represented by a combination of a resource and a sensor; (b) an instance of a rule that turns on the lights when motion is sensed. Note that this specific rule has no condition associated with it (images based on Vanderhulst, 2010).

ReWiRe provides a JavaScript code editor to easily add rules at runtime, as shown in Figure 7.6. In this specific example, the light sensor of a SunSPOT mote[1] is used to control one of the lights in the smart environment. When executed, the JavaScript code inserts two ECAA^{-1} rules into the behaviour model at runtime. These rules are then automatically triggered whenever the light sensor readings of the SunSPOT mote change, and depending on their value, either turn the light on or off.

sunspot-light.js

File

```
1  importPackage(Packages.org.edm.service.light.proxy);
2  importPackage(Packages.org.edm.service.sunspot);
3
4  var light;
5
6  this.onLoad = function() {
7      onSetup();
8      if (light == null) {
9          proc.out.println("No light selected, exiting...");
10         proc.exit();
11     }
12     var evt = new BEvent("*", LightSensor.URI, "SunSPOT light sensor");
13     var condDark = new BCondition("isDark", "Is darkness sensed?");
14     var condLight = new BCondition("isLight", "Is lightness sensed?");
15     var actOn = new BAction("turnOn", "Turn light on");
16     var actOff = new BAction("turnOff", "Turn light off");
17
18     proc.addRule(new BRule(evt, condDark, actOn, actOff));
19     proc.addRule(new BRule(evt, condLight, actOff, actOn));
20 }
21
22 this.isDark = function(ctx) {
23     proc.out.print("isDark?");
24     var s = new LightSensor(); s.fromContext(ctx.getParameters());
25     var b = (s.getValue() <= dt);
26     proc.out.println(" " + b + ", " + s.getValue());
27     return b;
28 }
```

Figure 7.6: Script editor for adding behaviour rules using a few lines of JavaScript.

1 http://www.sunspotworld.com/

7.4.2 *Annotating ReWiRe's Behaviour Model*

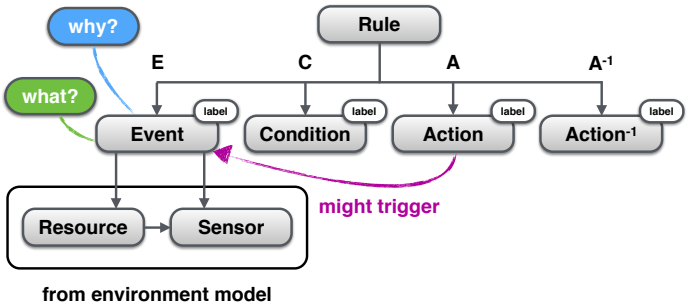
The way ReWiRe responds to context changes is encapsulated in its behaviour model, which makes it our main source of information for generating 'why?' and 'why not?' not questions. Although ReWiRe allows us to easily track sensor events and triggered behaviour rules, there is still some information missing to be able to easily generate why questions. First, we need human-readable descriptions of events, actions, and conditions that can be transformed both into statements (e.g., "the light turned off") and questions ("Why did the light turn off?"). Secondly, to be able to support 'why not?' questions, we need to know what would happen when a rule executes (e.g., possibly change the state of a light). Finally, it is important to take into account that users might themselves have an influence on the state of the environment model. For example, in Figure 7.2 *B.4*, Bob might use the control interface to turn on the light, and thus change that light's state. When he would later ask why the light was on, the system should be able to tell him that it was because of his own action in the control interface.

7.4.2.1 *Semantic Annotations*

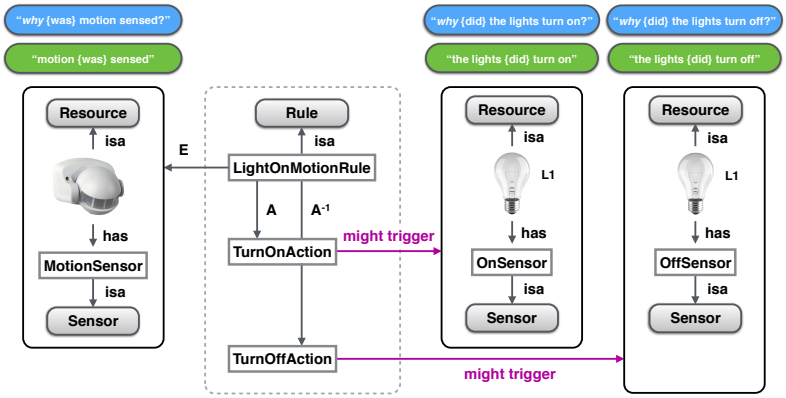
We extend ReWiRe's behaviour model with a number of additional properties and relations to be able to generate why questions and answers, as shown in Figure 7.7a. All events, actions and conditions are expected to have short descriptive labels, and events that directly affect end-users are enriched with *what* and *why* descriptions. These are plain text strings in which grammatical constructs such as auxiliary verbs are annotated to easily generate both negative and positive forms of a question or answer (e.g., "Why did ...?" and "Why didn't ...?").

The behaviour model was further extended with the *might trigger* relation between actions and events, which indicates what events can possibly be triggered when executing a rule. By adding this relation we not only know what events trigger a certain rule, but also what events (could) result from executing that rule. This information is essential both for answering 'why?' questions and for generating a list of 'why not?' questions (see Section 7.5). Figure 7.7b shows an annotated version of the motion sensor rule in Figure 7.5b. Note that we add both 'why' and 'what' descriptions for the motion event that triggers the rule and for the 'light on' and 'light off' events. Both the action and inverse action are connected to their respective light event using the 'might trigger' relation.

Developers can easily add annotations in the JavaScript behaviour rule editor when creating new rules (both at design-time and at runtime), as shown in Figure 7.8. In this example, we add one of the rules from the scenario that turns off the lights whenever a movie is playing (see Section 7.2). Note that by calling the *addMightTrigger* method (line 7), we indicate that the rule's action might cause the *LightOffSensor* event to be fired.



(a) Annotations to ECAA^{-1} rules



(b) An annotated version of a rule

Figure 7.7: Annotations added to ReWiRe's behaviour model: (a) short descriptive labels for each event, condition, and (inverse) action together with 'what' and 'why' descriptions for events; (b) an annotated version of the rule from Figure 7.5b, including the 'might trigger' relation.

```
1 importPackage(Packages.environment.behavior.model);
2 importPackage(Packages.services.light.sensors);
3
4 var evt = new BEvent("*", MoviePlayingSensor.URI,
5                      "Movie started playing");
6 var act = new BAction("turnOff", "Turn light off");
7 act.addMightTrigger("*", LightOffSensor.URI);
8
9 proc.addRule(new BRule(evt, act));
10
11 this.turnOff = function() {
12     alert("Turning off light");
```

Figure 7.8: Annotations can be easily added in the JavaScript behaviour editor.

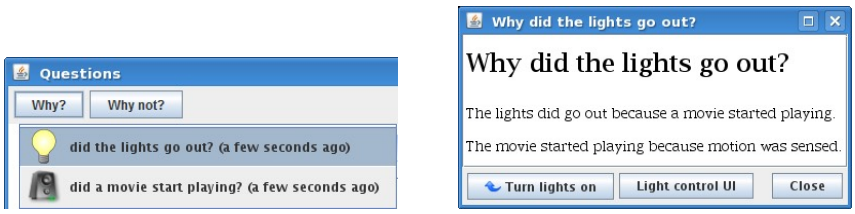
7.4.2.2 *Logging user actions*

As mentioned before, context-aware ubicomp environments typically consist of a mix of user-driven and system-driven behaviour. It is therefore important that we can also explain when changes to the environment's configuration were caused by user input. To achieve this, control user interfaces (e.g., Figure 7.2 *B.4*) log every user action. We track user actions by creating underlying behaviour rules whenever the user interacts with a control interface, which, for example, allows us to explain that the lights were turned off because the user did this in the lights control user interface. Analogous explanations could be provided when users make changes in other control user interfaces, such as the media control interface in Figure 7.3 (*D*) on page 131. We can also define inverse actions for these rules to allow users to revert to the previous state. Combining explanations for both system and user actions provides insight into how the system reached its current state.

7.5 SUPPORTING WHY QUESTIONS AND PROVIDING CONTROL

7.5.1 *Generating Questions*

The why questions menu (Figure 7.9a) is available at runtime and lists questions about events that are relevant to the user. It is not tied to any specific application, making it an example of an *external* intelligibility and control user interface (see Section 3.3.3). As shown in Figure 7.9a, questions are listed in a pop-up menu, together with a representative icon and the time at which they occurred. Remember that the list of questions includes both system and user actions and is sorted in reverse chronological order (most recent first).



(a) The why menu.

(b) An answer to a why question.

Figure 7.9: The why menu allows users to pose why and why not questions about events that happened in the environment. Users receive answers to their questions, and are offered a means to recover from undesired behaviour.

PervasiveCrystal automatically generates the 'why?' questions menu by monitoring the behaviour model. Any triggered event with a 'why' and 'what' description is transformed into a question string and added to the menu. Question strings are created from their 'why' descriptions. As mentioned before, grammatical constructs have been annotated in these descriptions to allow us to easily generate both 'why?' and 'why not?' forms of the question. Developers can exclude events from showing

up in the menu by not adding 'why' and 'what' descriptions for these events, which can be useful for internal events that are less relevant to end-users.

The 'why not?' menu includes questions about events that did not happen. To generate a list of 'why not?' questions, PervasiveCrystal analyses the 'might trigger' annotations. Recall that these annotations provide information about which events might possibly be triggered when a certain rule would be executed. There are two situations for which PervasiveCrystal adds a 'why not?' question. First, we add a question when an ECA rule is triggered but fails to execute because its condition did not evaluate to true. Note that a 'why not?' question about this ECA rule asks about the event that would be triggered as a result of executing its action. Secondly, rules might also fail to execute because their triggering event (E) did not occur regardless of the rule's condition, as in Figure 7.3. PervasiveCrystal also provides the option to list 'why not?' questions about those rules by analysing all rules and their 'might trigger' annotations. We decided to keep the 'why not?' menu separate from the 'why?' menu in order to avoid cluttering the menu with numerous 'why not?' questions in case the system consists of a large number of rules.

7.5.2 *Generating Answers*

When a user selects one of the questions in the pop-up menu, an answer dialogue is shown, as seen in Figure 7.9b. While we use 'why' descriptions to generate questions, 'what' descriptions are used for generating answers to questions (see Figure 7.7). Additionally, we use the 'might trigger' relation in the annotated behaviour model to provide answers. In the scenario of Section 7.2, Bob poses the question "Why did the lights go out?", and is shown a corresponding answer dialog (Figure 7.9b). The *what description* of this particular event would then be "the lights did go out". This description is used by the system to provide the first part of the answer, as shown in Figure 7.9b. The rest of the answer is automatically generated by analysing the ECA rules. The event about which Bob asks a question is traced back to the ECA rule that caused that event to happen using the 'might trigger' relation (see Figure 7.2 *scene 3*) and the 'what' description of that rule's triggering event is added to the answer.

PervasiveCrystal looks for the responsible rule by querying the behaviour model for recently executed actions that have a 'might trigger' annotation for the event the user posed a question about (see Figure 7.7). It is important to note that to ensure optimal performance and flexibility, ReWiRe's architecture does not allow us to trace these events across several distributed components. Consequently, there is some uncertainty involved in determining which action caused the event to happen. We currently use a 1-second time window to filter out subsequent similar events that might have been caused by other rules.

When PervasiveCrystal finds a rule that could be responsible for the event, it completes the answer by explaining what event caused that rule to execute. The responsible event's *what* description is added to the answer (in this case: "because a movie started playing'), as seen in Figure 7.9b. When the responsible event resulted from another rule, PervasiveCrystal repeats the process for that rule as well and

adds its corresponding answer below the previous one. In Figure 7.9b, the system adds a new paragraph with the explanation of why the movie started playing. When an executed rule required a condition to be true, this condition is also added to the explanation using its short descriptive label. If the chain of ECA rules eventually traces back to a user action that was performed in a control user interface, the explanation would describe this with "because you did . . . ".

Answers to *why not* questions are generated in a similar way, but require more work as there is more uncertainty in answering these questions. Since a 'why not?' question asks about an event that never occurred, the system cannot rely on tracing events and has to reason about what could happen. There are typically much more possible answers to a 'why not?' question than to a 'why?' question.

As mentioned before, PervasiveCrystal supports 'why not' questions in two ways. First, why not questions are possible about events resulting from rules which fired but did not execute because their condition was false. The corresponding answer will then explain that the event did not occur because the condition did not hold. Secondly, we allow why not questions about events resulting from rules which never executed because their triggering event never fired. In this case, the answer will explain that the event did not occur because the event that should have triggered the rule never occurred. We currently use a brute force approach to answer the second type of 'why not?' questions, where PervasiveCrystal enumerates all possible causes. As a next step, we would like to include a better estimation of timeliness (related to the type of event) and the likelihood of the candidate answers.

7.5.3 *Providing Control*

Once users have understood why the system acted in a certain way, they can choose to intervene and correct the system, if necessary. PervasiveCrystal supports two control mechanisms: *undo/do* and *control user interfaces*.

The *undo* operation is supported by calling the inverse action (A^{-1}) of an $ECAA^{-1}$ rule (see Section 7.4 and Figure 7.5a). The *do* operation, on the other hand, will just execute the action (A) of an $ECAA^{-1}$ rule, regardless of its event or condition. *Undo* is available in answer dialogues for 'why?' questions, while *do* is available in answer dialogues for 'why not?' questions. In Figure 7.9b, Bob asked why the lights were turned off. The button on the left allows Bob to turn the lights back on, thereby *undoing* the action taken by the system (as indicated by the button's icon).

Besides *undo* and *do*, we also provide more fine-grained control user interfaces to intervene and correct the behaviour of the system. As shown in Figure 7.9b, there is a second button on the right that invokes the light control user interface as seen in Figure 7.2 (*B.4*). ReWiRe supports additional annotations for behaviour rules that indicate the different goals those rules contribute to (e.g. controlling lights, playing media). When generating answer dialogues, PervasiveCrystal analyses the annotations for those rules, and creates a specific button that brings up the corresponding control user interface for that goal.

As mentioned before, we log all user actions by creating underlying behaviour rules for those actions. We also apply this technique to both control mechanisms:

all *undo* and *do* actions are logged, as well as user actions that are performed in control user interfaces.

7.6 USER STUDY

7.6.1 *Participants and Method*

We conducted a small, informal user study with PervasiveCrystal (Vermeulen et al., 2009b) to get an initial idea of the suitability of the technique and ease of use of the why questions interface. We asked five colleagues in our lab (4 male, 1 female) to use our system to understand and control the behaviour of a ubicomp environment in different situations. Four out of five participants were computer scientists, while the fifth participant had a background in social sciences and no programming experience. The experiment was carried out in a realistic ubicomp environment: a demo room for an interactive museum which next to museum artefacts also featured different kinds of sensors, as well as displays and speakers to provide visitors with information. Subjects used a networked Ultra-Mobile PC (a Samsung Q1 Ultra running Windows XP and the PervasiveCrystal client software) to ask why questions and view the corresponding answers, as shown in Figure 7.10.

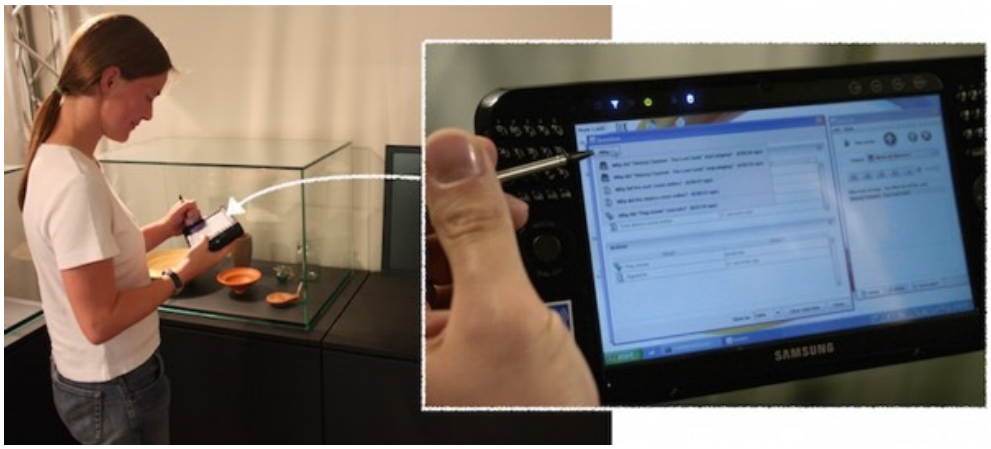


Figure 7.10: The setup for the study: participants used PervasiveCrystal on an UMPC in a smart museum environment.

Participants were presented with three situations in which something happened that they had to explain and control using PervasiveCrystal. All participants followed the same order of tasks. In the first task, participants were told to sign on to the system after which medieval music started playing in the room. For the second task, participants were asked to go and stand in front of a display that detected the user's presence using a webcam (and a very simple motion detection algorithm). When subjects would walk up to the display, a movie would start playing. The third and final task was similar to the second one, but here the movie would only play as

long as motion was being detected and would immediately stop otherwise. Participants were asked to try to understand what was different compared to the previous situation. Once they figured out how the system worked, we asked to use one of our control mechanisms to find a way to keep playing the movie without having to keep moving in front of the display. Finally, we conducted a semi-structured interview in which participants were asked to comment on the different features of PervasiveCrystal. The questionnaires used for the interview can be found in Appendix B.4. The experiment took about 50 minutes in total for each participant (20 minutes for the tasks, 30 minutes for the interview).

7.6.2 *Observations*

Each participant agreed strongly that 'why?' and 'why not?' questions are useful to allow users to understand what happens in their environment and offer them control over this behaviour. All subjects were able to use the questions interface to find the cause of events in these three tasks. Overall, participants found that the answers to the 'why?' and 'why not?' questions were what they wanted to know, although most participants argued that the way they were presented could be somewhat improved.

A problem that most users faced was the fact that the why-menu quickly became cluttered when many events were firing in a short time span. This made it hard for subjects to find the question they wanted to ask. Especially in the third task, users would several times trigger the motion sensor, resulting in a large number of why questions in the user interface. There are a couple of ways to overcome this problem. First, users could be offered a way to filter events (e.g., only show questions about music, video, or lights). Secondly, events that occur very often in a short period of time could be clustered into a single why question (e.g., "Why did a movie stop playing (10x in the last 20 seconds)").

Participants experienced some more difficulties with the control mechanisms. Three out of five subjects were able to successfully use our control mechanisms without assistance, and achieved the desired effects . The remaining two participants had to be given a few cues, but did not require much assistance to complete the tasks. Participants mostly preferred to use the fine-grained control user interface, such as the light control user interface in Figure 7.2 (*B.3*–*B.4*). Subjects found this mechanism useful and could quickly figure out how to use it to control the environment. They were less positive about the ease of use of our *undo* and *do* commands. In the first iteration of PervasiveCrystal that we used during the study, we used generic labels ("undo" and "do") for these commands, which made it hard for users to predict their effect. We later changed our implementation to use more specific labels for the undo and do buttons based on the label for the corresponding action, such as "Stop video" and "Turn on lights".

Finally, one participant mentioned that although he found it useful to pose why questions, he felt they were mainly appropriate for understanding and controlling the environment "after the facts". He argued that there should be other mechanisms that allow users to see what will happen before an action will be executed, and

offer them the means to prevent it from being executed. As discussed earlier, why questions are indeed mostly useful to understand or revert past behaviour, while techniques such as feedforward (Chapter 5) or slow-motion feedback (Chapter 6) can be used to provide intelligibility and control before and during execution of actions.

7.7 LIMITATIONS AND POSSIBLE EXTENSIONS

In this section, we provide a brief overview of the limitations of PervasiveCrystal and possible extensions.

7.7.1 *Scalability*

It is yet unclear whether the system can scale to large and complex applications. Especially 'why not?' questions can pose problems since the number of possibilities that have to be examined for these questions increases rapidly with the complexity of the system. It will be necessary to find a balance between reasonable memory usage and performance on the one hand, and sufficiently detailed information on the other hand. Nevertheless, we believe the biggest challenge in scaling PervasiveCrystal to large and complex applications lies not in optimizing CPU or memory usage, but rather in providing users with detailed and complete information about the system's behaviour without leaving them overwhelmed.

7.7.2 *Support for Machine Learning*

PervasiveCrystal currently only supports rule-based context-aware systems. Unlike the intelligibility toolkit (Lim and Dey, 2010), it was not designed to deal with machine learning algorithms such as decision trees, Hidden Markov Models, support vector machines or neural networks. Next to being able to provide explanations for machine learning algorithms, it would be useful to support learning from user's interactions with the system. When users repeatedly *undo* certain actions, for example, PervasiveCrystal could then automatically suggest a modification to that rule corresponding to the user's desired behaviour, as done in the IOS system (Cheverst et al., 2005).

7.7.3 *Supporting Other Types of Questions*

Although we currently only support 'why?' and 'why not?' questions, our behaviour model could also be used to answer other types of questions. Lim and Dey (2009) explored users' demand for five types of questions (including 'why?' and 'why not?' questions). We feel that especially their 'what if?' and 'how to?' questions could be useful additions to PervasiveCrystal. Both questions help to address the gulf of execution: 'what if?' questions explain to users how the system will respond given a certain event or condition—which is a way to support feedforward (Chapter 5)—

while 'how to?' questions help users to understand how they can achieve a certain effect. The annotated behaviour model that we use for why questions such as the 'might trigger' relation (Figure 7.7), can also help us to answer these other two questions. By linking the event (E) and condition (C) of all ECA rules to the events that their actions might trigger, we could come up with a reasonable guess of what events can result in a certain desired effect, and thus answer 'how to?' questions. That same chain of events can be used to generate answers to 'what if?' questions, although a filtering mechanism might be necessary as an event could have a multitude of possible effects. Although Lim and Dey (2010) explored different explanation strategies, more research is needed to determine a suitable user interface for posing and answering these questions, since textual explanations might become complex and hard to understand. For performance reasons, it might be necessary to limit the nesting level while creating the chain of events, as the set of events that can possibly be triggered will rapidly increase with each iteration.

7.8 CONCLUSION

In this chapter, we explored allowing users to pose 'why?' and 'why not?' questions about the behaviour of context-aware applications and environments. With respect to the timing dimension in our design space (Section 3.3.1), this is a way to explain the system's reasoning and provide intelligibility and control *after* the action. We presented PervasiveCrystal, a framework to easily develop context-aware applications that support asking and answering 'why?' and 'why not?' questions. The foundation for our approach is a flexible behaviour model that can be easily queried to automatically generate both questions and answers. We also explored the suitability and ease of use of this technique in an initial user study.

8

INTELLIGIBILITY AND CONTROL FOR PROXEMIC INTERACTIONS

8.1 INTRODUCTION

In this chapter, we describe a *case study* in supporting intelligibility and control for *proxemic interactions*. Proxemic interactions feature people-aware ensembles of devices that employ fine-grained knowledge of the identity, proximity, orientation or location of their users, which is essentially a specific type of context information. We discuss the design and implementation of *dynamic peripheral floor visualizations* to address interaction challenges with proxemic-aware interactive surfaces. Our *Proxemic Flow* system uses a floor display that plays a secondary, assisting role to aid users in interacting with the primary display. The floor informs users about the *tracking status*, indicates *action possibilities*, and *invites* and *guides* users throughout their interaction with the primary display. We believe this *case study* can serve as inspiration to designers looking to support intelligibility and control in different ubicomp applications.

In the remainder of this section, we introduce the idea of proxemic interactions, clarify the relevance of interaction challenges with proxemic interactions to this dissertation and provide an overview of our *Proxemic Flow* approach.

8.1.1 *Proxemic Interactions*

The notion of *proxemic interactions* was introduced by Greenberg et al. (2011) as a new direction for the field of ubiquitous computing (Weiser, 1991). Proxemic interactions features people-aware ensembles of devices that employ fine-grained knowledge of the *identity*, *proximity*, *orientation* or *location* of their users, as shown in Figure 8.1.

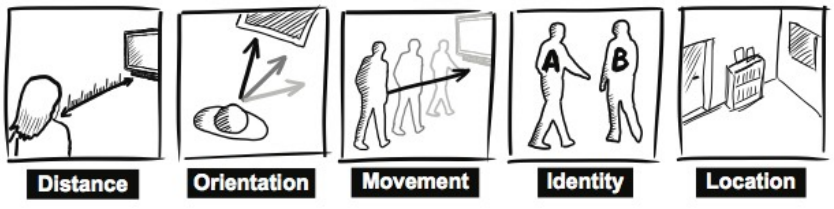


Figure 8.1: The five dimensions of proxemics (image source: Greenberg et al., 2011).

Proxemic interactions is based on anthropologist Edward T. Hall's *theory of proxemics* (Hall, 1969), which investigated the use of distance in nonverbal communica-

tion. In particular, proxemics theory identified the culturally-specific ways in which people use interpersonal distance to understand and mediate their interactions with others. Proxemics is is not limited to interpersonal communication, but also extends to "the organization of space in [our] houses and buildings, and ultimately the layout of [our] towns" (Hall, 1963). Indeed, in the vision put forward by Marquardt, Greenberg, and colleagues (Greenberg et al., 2011; Ballendat et al., 2010; Marquardt et al., 2012a,b), proxemic relationships are used to mediate interaction between people and *ensembles* of different digital devices, such as mobile devices or large interactive surfaces. Additionally, they envision devices to take into account the non-digital, semi-fixed or fixed objects in the user's physical environment (Greenberg et al., 2011).

One of the most commonly featured aspects of Hall's theory in HCI is the use of four *proxemic zones* that correspond to our interpretations of interpersonal distance: the *intimate*, *personal*, *social*, and *public* zone (Greenberg et al., 2011). In earlier research, these different interaction zones have been used to mediate interaction with large interactive surfaces (Prante et al., 2003; Vogel and Balakrishnan, 2004; Ju et al., 2008). The notion of proxemics has also been used before to facilitate cross-device interaction (Hinckley et al., 2004; Hinckley, 2003; Gellersen et al., 2009; Kray et al., 2008).

In recent years, large interactive surfaces such as vertical displays or tabletops, are increasingly appearing in semi-public settings (Brignull and Rogers, 2003; Ojala et al., 2012). With the availability of low-cost sensing technologies (e.g., IR range finders, depth cameras) and toolkits such as the Proximity Toolkit (Marquardt et al., 2011) or the Microsoft Kinect SDK[1], it is fairly easy to make these large displays react to the presence and proximity of people, which has been picked up both by researchers (e.g., Müller et al., 2009a, 2012; Jurmu et al., 2013) and commercial parties (see Greenberg et al., 2014 for several examples). Although these low-cost sensing solutions tend to apply fairly crude measures of proxemics and only take into account a few proxemic dimensions (see Figure 8.1), it does mean that proxemic interactions is becoming more commonplace in our everyday environments.

8.1.2 *Relation to Context-Aware Computing and Relevance to This Dissertation*

Proxemic relationships between devices and people are essentially just a particular type of *context information* (Dey et al., 2001) that applications can take into account. In fact, early research in context-awareness already explored interaction with systems that are aware of the user's location and identity. Indeed, Schilit et al. (1994) describe context-aware systems as systems that adapt to "the collection of *nearby* people, hosts and accessible devices" [emphasis added], and define three important aspects of context: "where you are, who you are with, and what resources are nearby". Schilit et al. also describe a number of interaction techniques such as *proximate selection*, where nearby objects are easier to select (e.g., nearby printers).

1 http://www.microsoft.com/en-us/kinectforwindows/

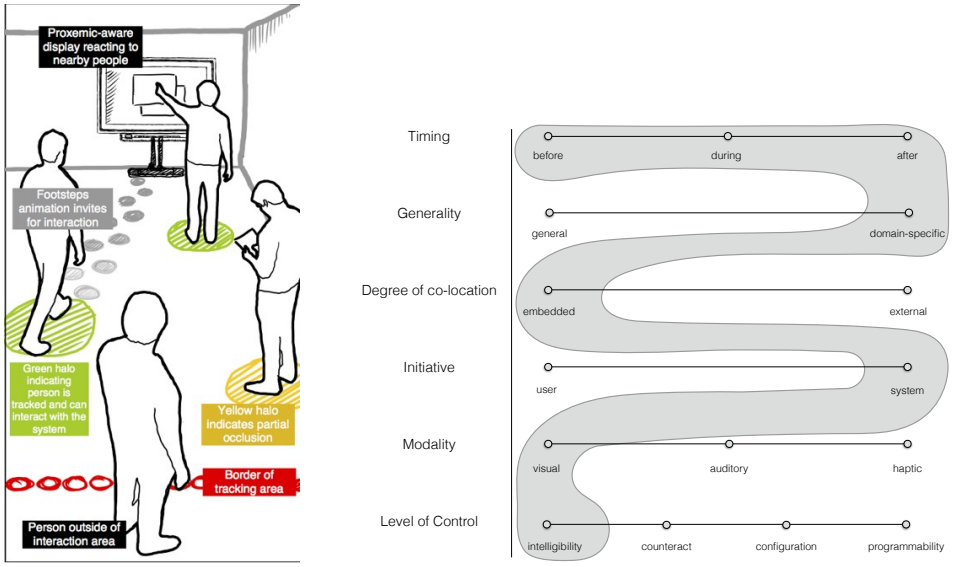
Since proxemic-aware systems are thus a particular type of context-aware systems, they can also suffer from interaction challenges similar to the ones that we address in this dissertation—such as issues in dealing with systems that rely on implicit interaction (see also Section 2.3). Greenberg et al. (2011) talk about a number of open challenges for proxemic interactions such as designing suitable *behaviour rules* that dictate how a proxemic entity responds to both implicit and explicit input, and how it can be controlled. In fact, the use of implicit interaction can also be deliberately exploited by designers to *trick* users into performing certain actions that do not have their best interests in mind (Boring et al., 2014). Some people-aware advertising displays in public spaces, for example, make it very difficult for users to opt-out of the interaction. In addition, several studies have reported problems encountered by users when interacting with public displays, such as *display blindness* (Müller et al., 2009b; Huang et al., 2008) when people fail to notice the display, or *interaction blindness* (Ojala et al., 2012; Houben and Weichel, 2013) when people fail to notice that the display is interactive. As stated by Müller et al. (2010), the commonly used interaction modalities for these displays—e.g., proximity, body posture, mid-air gestures—are hard to understand at first glance. In another study, Jurmu et al. (2013) reported that users had difficulties pinpointing the exact zone where the display would react to their input, especially when the display was also reacting to the input of other people.

In summary, we argue that—as with context-aware systems in general—proxemic interactions need to *support intelligibility and control*. The complexity of proxemic interactions—with its reliance on implicit interaction and heavy use of rich spatial relationships between multiple people and devices—makes it an interesting application domain for this dissertation research. The focus of this chapter will therefore be on exploring how to provide support for intelligibility and control in systems that make such heavy use of spatiality and implicit interaction.

8.1.3 *Proxemic Flow: In-Situ Floor Visualizations to Mediate Large Surface Interactions*

In this chapter, we present the design and implementation of dynamic, peripheral floor visualizations as a means to address interaction challenges with large interactive surfaces. The core aspect of our approach is the notion of combining a vertical interactive display with a *secondary, peripheral floor display*. In contrast to existing work on interactive illuminated floors (e.g., Augsten et al., 2010; Schmidt et al., 2014), our LED floor display is not used as a primary interaction space, but plays a secondary, assisting role to aid users in interacting with the main display. As shown in Figure 8.2a, our *Proxemic Flow* system uses the floor to inform users about the tracking status, to indicate action possibilities, and to invite and guide users throughout their interaction with the primary interactive surface.

Proxemic Flow can be situated in the design space that was introduced in Chapter 3, as shown in Figure 8.2b. With respect to *timing*, the in-situ floor visualizations provide information to users that is relevant before, during and after actions. In section Section 8.4, we explain in detail when visualizations are shown (see also Table 8.1 for an overview). Since it is specifically oriented towards addressing inter-



(a) (b)

Figure 8.2: (a) Proxemic Flow provides awareness of tracking and fidelity, action possibilities, and invitations for interaction; (b) the shaded region shows where Proxemic Flow fits in the design space for intelligibility and control.

action challenges in proxemic interactions, Proxemic Flow is *domain-specific* (unlike e.g., feedforward). The visualizations are integrated with the application on the primary display and are shown in the space around the large interactive surface, making Proxemic Flow an *embedded* technique. The floor display also allows for providing location-specific information, such as inviting users to specific interaction zones using the footsteps visualization (as shown in Figure 8.2a). The *initiative* lies with the system, as floor visualizations are shown automatically. The floor display currently uses only *visuals* to convey information to users, although this could also be combined with other modalities in the future. Finally, Proxemic Flow offers no explicit control to users besides making them aware of e.g., tracking and opt-in and opt-out areas. Based on this information, users can regulate their own behaviour (e.g., move out of the tracking area). Regarding the *level of control*, Proxemic Flow thus only offers intelligibility.

In the future, we imagine these floor displays to be integrated in public spaces to accompany vertical public displays. Indeed, due to their relatively low cost, LED floor displays are already fairly common in urban spaces (Figure 8.3). For example, Figure 8.3a shows flashing LED lights that show arriving trains in Washington DC metro stations.

In what follows, we first provide a brief overview of the different interaction challenges with proxemic interactions that we attempt to address with our technique, followed by a discussion of related work. We then explain our in-situ floor visualization strategies with a running example application, inspired by the design of

(a) LEDs show arriving metro trains in Washington DC[2]

(b) Displays integrated into a staircase at the Lincoln Center, New York City[3]

Figure 8.3: Examples of the use of LED floor displays in urban spaces.

the Proxemic Media Player (Ballendat et al., 2010). Next, we discuss our floor visualization architecture in detail. Finally, we discuss different design considerations that are important for in-situ floor visualizations, summarize our contributions, and conclude with opportunities for future work.

8.2 BACKGROUND AND MOTIVATION

Previous studies have revealed several issues that users face when interacting with large interactive surfaces. In this section, we provide an overview of the core interaction challenges that motivated our work.

8.2.1 *Intelligible Sensing*

A common problem users experience while interacting with public displays is a lack of feedback about how the system is currently recognizing and interpreting input from the user, and how reliably the user's input is being sensed—which might explain interaction problems. In addition, users might not realize that the display is interactive (Ojala et al., 2012), and might be surprised when it suddenly starts reacting to their presence. Tracking feedback could inform users that they are recognized by the display, and how it is interpreting their movements.

2 https://www.youtube.com/watch?v=DppgBi0ZMc8
3 https://www.youtube.com/watch?v=BzLicqgh3fQ

8.2.2 *Implicit Interaction*

One of the core reasons for interaction challenges with proxemic-aware interactive surfaces is their reliance on implicit interaction. For example, the Proxemic Media Player (Ballendat et al., 2010) automatically pauses videos when two people are both oriented away from the display (e.g., when starting a conversation), which might be very surprising and perhaps disturbing when users first encounter such a system. Indeed, Ballendat et al. (2010) argue that defining the rules of behaviour that indicate how systems using proxemic interactions interpret and react to users' movements is a key question to be addressed. We argue that it is very important to not only inform users of how they are being tracked by the system, but also to indicate how the system is taking action (or about to take action) based on their movements. For example, pausing the video in the previous example could be made clear using *slow-motion feedback* (Chapter 6).

8.2.3 *Invisibility of Action Possibilities and Lack of Guidance*

Users typically have difficulties knowing how they can interact with a large display. As stated by Müller et al. (2010), the commonly used interaction modalities for public displays (e.g., proximity, body posture, mid-air gestures) are often hard to understand at first glance. For example, when the display reacts to the user's location in different interaction zones (Vogel and Balakrishnan, 2004), the invisibility of these zones causes problems with identifying the exact zone where the display reacts to their input, especially when the display is also reacting to the input of other people (Jurmu et al., 2013). Next to showing the possible actions that users can perform, *feedforward* (Chapter 5) might be useful to inform users of what will happen, for example, when approaching the display.

8.2.4 *Lack of Support for Opt-in and Opt-out Mechanisms*

Another problem is the lack of explicit opt-in or opt-out mechanisms, which is especially important in (semi-)public spaces. Jurmu et al. (2013) and Brignull and Rogers (2003) found that users sometimes want to avoid triggering and just passively observe the display. In addition, O'Hara (2010) reports that users sometimes perform actions that were not meant to be interactive, but were merely social gestures. Greenberg et al. (2014) further discuss how interactive surfaces in semi-public settings typically lack opt-in and opt-out choices (either deliberately or unintentionally). They state that, at the very least, a way to opt-out should be provided when people have no desire to interact with the surface. Furthermore, users might want to know what will happen when they leave or opt-out. Will the surface be reset to its original state, and what will happen to their personal information still shown on the screen?

8.3 RELATED WORK

8.3.1 *Feedback, Discoverability and Guidance for Large Interactive Surfaces*

Existing work has explored techniques that aim to address the previously introduced interaction challenges for large interactive surfaces. Here, we give a brief overview of related techniques, categorized by the challenges that they address.

8.3.1.1 *Attracting Attention to Overcome Display Blindness and Interaction Blindness*

Existing work has explored how displays can attract attention to motivate users to interact with a display, and thus overcome *display blindness* (Müller et al., 2009b) and *interaction blindness* (Ojala et al., 2012). Researchers have identified different barriers (Brignull and Rogers, 2003; Michelis and Müller, 2011; Cheung et al., 2014) that prevent users to transition from merely passing by to actively interacting with a public display. For example, the Audience Funnel (Michelis and Müller, 2011) categorizes interaction with public displays in six different interaction phases, where barriers (or 'tresholds') prevent users from transitioning from one phase to the next. Müller et al. (2010) describe a number of techniques that can be used to attract attention, such as curiosity and exploration, fantasy and metaphor, or surprise. Seto et al. (2012) and Cheung and Scott (2013) explored the use of animations to convey interactivity and improve discoverability. Houben and Weichel (2013) found that placing a curiosity object near the display that is visibly connected to it, significantly increased interactivity with the display. Others have explored the use of mobile devices to overcome interaction barriers with public displays (Holleis et al., 2007; Cheung et al., 2014). As a more extreme example, the Proxemic Peddler (Wang et al., 2012) continuously monitors the user's distance and orientation and even attempts to reacquire the attention of a passer-by when they turn or move away from the display.

With a few exceptions (e.g., Houben and Weichel, 2013), most techniques attempt to convey interactivity and attract users by using visualizations on the display itself. In contrast, with Proxemic Flow, we rely primarily on the floor display to convey interactivity. One of the advantages of providing visualizations on a secondary display, is that they do not occlude or distract from existing content on the primary display. The floor can explicitly reveal the interaction area through borders and zones, and shows halos whenever people are recognized by the display. In addition, our focus lies not on drawing as many people in as possible and maximizing the number of people that interact with the primary display. Rather, our floor visualizations are designed to make it clear that passers-by *can* interact, make them aware that they are tracked, and also help people avoid interacting with the display, if they wish to do so.

8.3.1.2 *Revealing Action Possibilities and Providing Guidance*

A number of systems for large interactive surfaces suggest action possibilities and input mechanisms by visualizing sensor data, such as depth camera images, de-

tected user skeletons (Jurmu et al., 2013; Grace et al., 2013; Beyer et al., 2014) or mirror images (Müller et al., 2012). Early work by Vogel and Balakrishnan (2004) already included a self-revealing help feature using a mirror image video sequence that demonstrates the available gestures. Walter et al. (2013) studied different on-screen visualizations to reveal a 'teapot' gesture that allows users to indicate that they would like to start interacting with the display using mid-air gestures. Grace et al. (2013) investigated different visualizations to convey interactivity of a large public display, including skeleton visualizations, and a combination of a skeleton and a spotlight. In addition, a number of gesture guides have been developed for horizontal tabletop surfaces (e.g., Freeman et al., 2009; Vanacken et al., 2008).

Although techniques such as these are useful for revealing action possibilities—e.g., (mid-air) gestures and body postures—on the display itself, we feel that in-situ techniques can be more appropriate for proxemic-aware surfaces. After all, proxemic interactions make extensive use of spatial movements around people and devices, which can be difficult to visualize on a single 2D display. Techniques that reveal these action possibilities on the primary display tend to focus mostly on one or a few proxemic dimensions. For example, cross-device interaction techniques typically use *orientation* around the user's device to show possible other target devices, as with the Gradual Engagement pattern (Marquardt et al., 2012a) or the Relate system (Gellersen et al., 2009).

8.3.1.3 *Providing Tracking Feedback*

A number of systems have explored showing tracking feedback for proxemic interactions to convey what the system 'sees' of the user. As argued by Bellotti et al. (2002), this information is essential to help users interact with sensing systems. As mentioned before, several systems visualize detected skeletons (e.g., Beyer et al., 2014) or show mirror images (e.g., Müller et al., 2012). With their interactive whiteboard, Ju et al. (2008) showed a dot pattern visualization to indicate in which proximity zone the user was recognized. In the Medusa proximity-aware tabletop (Annett et al., 2011), the user's proximity is visualized using an orb visualization. Both in the Proxemic Media Player (Ballendat et al., 2010) and with the Gradual Engagement pattern (Marquardt et al., 2012a), tracked devices are visualized on the large surface with their relative size mapped to their proximity to the large display.

Again, the main difference with Proxemic Flow, is that our visualizations provide in-place tracking feedback on the floor, in the space where users are tracked. Earlier work also used halos to provide tracking feedback on floor displays. Hespanhol et al. (2014) describe the Solstice LAMP, an installation that uses a projected floor display as a proxy for interacting with a large media facade for interactive musical composition. Luminous shapes were projected around people, and would be merged and grow when multiple people would stand close together. Proximity Lab (Karatzas, 2005) is an installation that uses slippers equipped with RFID tags to track people across a floor display. This installation played sounds when people moved across the floor and when people's halos touched. Each user is represented by a different colour halo, whose colours get mixed when users interact. Although

these examples use halos to provide feedback about the current state of tracked entities (e.g., a person's tracked location), they do not convey information about the tracking accuracy.

8.3.1.4 *In-Situ Feedback and Guidance*

As mentioned before, proxemic-aware systems typically take different actions depending on the interaction zone in which the user is located (Ballendat et al., 2010; Ju et al., 2008; Vogel and Balakrishnan, 2004), which might be unintelligible to users. In earlier work, Rehman et al. (2005) visualized interaction zones in-place using an augmented reality technique. A disadvantage of their solution is that it requires users to wear a head-mounted display.

In later work, researchers explored the possibilities of providing in-situ feedback and guidance using a combination of projectors and (depth) cameras—thereby eliminating the need for users to wear additional apparel. For example, LightSpace (Wilson and Benko, 2010) shows when users are tracked by the system by projecting coloured highlights on the user's body, and can also indicate when users are recognized as a group. LightGuide (Sodhi et al., 2012) uses a projector and depth cameras to project visualizations on the user's body and provide movement guidance. Ozturk et al. (2012) explored people's reactions to projections of their 'future footsteps' in an airport terminal, a technique similar to our steps visualization (see Section 8.4.3).

Although the use of projectors allows for high-resolution visualizations and more flexibility, projectors still require low-lighting conditions, which makes these techniques less suitable for large interactive surfaces in urban spaces (especially during daytime). Regarding guidance using LED floors, Rogers et al. (2010) explored the use of LED lights embedded in carpet tiles that aimed to motivate people to use the staircase more often. Rogers et al. observed that the LED lights actually had a significant effect on people's behaviour, which illustrates that in-situ visualizations and guidance can be quite powerful.

8.3.2 *Interactive Illuminated Floors*

With Proxemic Flow, we propose the use of graphical information shown directly on the floor of the interactive space—all around the people interacting—for providing feedback about the system status or informing users of action possibilities and consequences. In this section, we briefly review the major applications for illuminated floors for interaction design and review technical implementations, before we introduce our novel in-situ floor visualization strategies in the following section.

Interactive illuminated floors have been used for several purposes, such as interactive art (Hespanhol et al., 2014; Karatzas, 2005) or games (Grønbæk et al., 2007), and have recently seen increasing exploration as a primary interaction space (Augsten et al., 2010; Bränzel et al., 2013; Schmidt et al., 2014). A variety of input and output technologies have been used for these interactive floors, such as tracking users through computer vision techniques (e.g., Grønbæk et al., 2007) or pressure

sensing (e.g., Bränzel et al., 2013), and showing output using projectors (Bränzel et al., 2013; Grønbæk et al., 2007), LED illumination (Dalton, 2013) or vibrotactile feedback (Visell et al., 2009). Our work extends this earlier research in the domain by (a) proposing the use of the floor as a peripheral/secondary output device that can help to mediate interactions with a different primary interaction device, and (b) providing a vocabulary of strategies to provide in-situ feedback about current and future interactions with the system.

8.4 IN-SITU FLOOR VISUALIZATION STRATEGIES

An important design challenge is providing learnability and discoverability, without overloading the UI. While additional feedback, such as showing the next possible actions, is necessary to guide users in the interaction, designers should avoid overwhelming users with too much information. We see large potential for guiding or inviting mechanisms that show users step-by-step which next actions are possible. In the context of learning gestures—another type of interface with similar challenges such as the invisibility of action possibilities—Wigdor and Wixon (2011) argue for adding a minimum of extra on-screen graphics to make gestures self-revealing, which they call 'just-in-time chrome'. Just-in-time chrome shows on-screen affordances to guide users to the next possible actions. We see similar techniques being useful for proxemic interactions. When a person moves in the space in front of the display, the system could, once the person is tracked and recognized, show possible other locations where they can move to, and possibly use animations to guide them towards these locations.

We introduce a series of in-situ visualizations on the floor that provide additional feedback to end-users interacting with the primary vertical display. These visualizations are categorized into three phases, progressing from:

(1) *in-situ tracking feedback*, answering the questions: "What does the system see? How well does the tracking work?"

(2) *revealing interaction possibilities*, answering: "What possible interactions are available?", and

(3) *inviting for and guiding interactions*, answering: "What can I do next?"

It is important to note that even though these floor visualizations bear some resemblance to techniques that were discussed earlier such as feedforward, they serve a different purpose. Rather than explaining what the result of an action will be, these visualizations are primarily targeted towards improving discoverability and conveying action possibilities, one of the important challenges that users face when interacting with large interactive surfaces (see Section 8.2). Nevertheless, there are a couple of subtle feedforward elements in the visualizations, such as the border visualizations that indicate where to go to opt-out of the interaction.

As mentioned before, we will illustrate our in-situ visualization strategies with an example application. Our photo gallery application (Figure 8.4) shows photo collections on a large public display. A series of interactions are possible with this gallery

application: it shows photo thumbnails when in idle mode, reveals more content when a person approaches the display, shows full screen photos when people stand directly in front of the display (or move back and sit down), and allows mid-air gestures to navigate the photo collection (e.g., waving with a hand left or right to browse through the timeline of photos). The application uses slow-motion feedback on the primary display (Chapter 6), as it continuously reveals more content when users get closer to the display (Figure 8.4b). While limited in scope, we believe that this example application captures the essence of many proxemic interactions applications (Greenberg et al., 2011) and is well suited to demonstrate our in-situ floor visualization strategies. Throughout this section, we will also refer to the use of our visualization strategies for other application contexts.

(a) from a distance (b) when moving closer

Figure 8.4: Our photo gallery application responds to the user's proximity to the display. When users approach the display, it will gradually reveal more thumbnails, a behaviour that is identical to the Proxemic Media Player (Ballendat et al., 2010).

Our visualization strategies are used for different purposes and have different properties. Table 8.1 shows an overview of our different strategies, indicates their purpose and to which of the three phases they correspond.

	Phase	Purpose	Perspective		Position		Temporal Relevance		
			Egocentric	*Exocentric*	*Static*	*Dynamic*	*Past*	*Present*	*Future*
Halos	1	Tracking feedback and quality	✔			✔		✔	
Trails	1, 3	Historic traces of action	✔	✔		✔	✔	✔ (a)	✔ (b)
Zones	2	Interaction zones/action possibilities	✔ (c)	✔	✔			✔	✔ (d)
Borders	2	Opt-in and opt-out indicators		✔	✔			✔	✔ (e)
Waves	3	Inviting for interaction		✔	✔				✔
Steps	3	Guiding spatial movement	✔			✔			✔

Table 8.1: An overview of our different floor visualization strategies.

The table compares our floor visualization strategies based on three different aspects: *perspective*, *position*, and *temporal relevance*. Regarding perspective, we distinguish between *egocentric* and *exocentric* visualizations. For example, tracking halos are targeted towards being viewed from the user's own perspective (egocentric),

while zones and borders are mostly useful from an external perspective (exocentric). Additionally, some visualizations have a *static* position on the floor, while others can move *dynamically* (e.g., together with the user). Finally, visualizations can be relevant to the user's current interactions with the primary display (the *present*) (e.g., quality of tracking), or can alternatively provide clues about *past* or *future* actions. We will now go over the three phases, and will later come back to these different aspects while discussing our floor visualization strategies.

8.4.1 *Phase 1. In-Situ Personal Tracking Feedback with Halos*

As discussed before, a fundamental challenge for the interaction with large surfaces is providing a person with immediate feedback about how the system is currently recognizing and interpreting gestures or other input from the user. This tracking feedback is an essential part of Bellotti and Edwards's definition of intelligibility (Bellotti and Edwards, 2001). In this section, we introduce visualization strategies to provide this feedback directly in the physical space in front of the display where the person is moving.

8.4.1.1 *Personal Halos*

The personal halo provides immediate feedback on the floor display about the tracking of a person in space. When the person enters the area in front of the public display, a green halo (with an area of approximately 1 m^2) appears underneath the person's feet (Figure 8.5a). The halo moves with them when moving in the tracking area, and therefore gives continuous feedback about the fact that the person is being recognized and tracked by the system.

Another important part of information (besides information about the fact that a person is tracked) is the actual quality of tracking. Most computer vision based tracking systems (RGB, depth, or other tracking) have situations where tracking works well, where it does not work well, or where it does not work at all (e.g., due to lighting conditions, occlusion, limited field of view). Therefore, our personal halo visualization encodes the quality of tracking in the colour of the halo. To indicate tracking quality, we use three different colours (Figure 8.5b–d). A green halo indicates optimal tracking of the person in space. Its colour changes to yellow when the quality of tracking decreases, for example when the person moves to the limits of the field of view or when partially occluded by another person or furniture. Finally, a red halo colour is shown when the tracking of the person is lost, such as when moving too far away from the camera, or if the occlusion is hiding the person completely. For this last case, since the person is now not tracked anymore, the red halo visualization remains static at the last known location of the person, fades in and out twice, and then disappears (the duration of that animation is approximately 4 seconds). If the person moves back into the field of view of the camera and the tracked region, the halo colour changes accordingly back to green or yellow.

The immediate feedback of tracking through halos can give a person the opportunity to intervene, e.g., to opt-out of interaction with the system. For example,

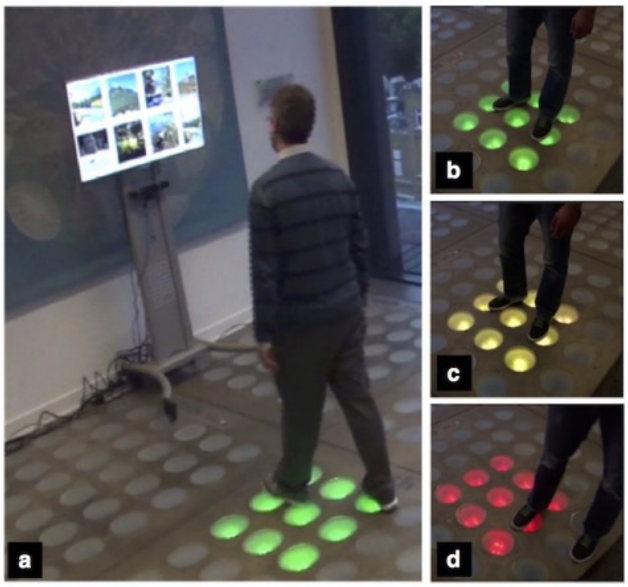


Figure 8.5: Halos provide feedback about active tracking (a), and also reveal the tracking quality: a green halo (b) indicates optimal tracking, a yellow halo (c) represents reduced accuracy, and a briefly pulsating red halo (d) shows that tracking is lost.

when noticing that they are being tracked, people could move back out of the active tracking area.

8.4.1.2 *Alternative Halo Visualization Strategies*

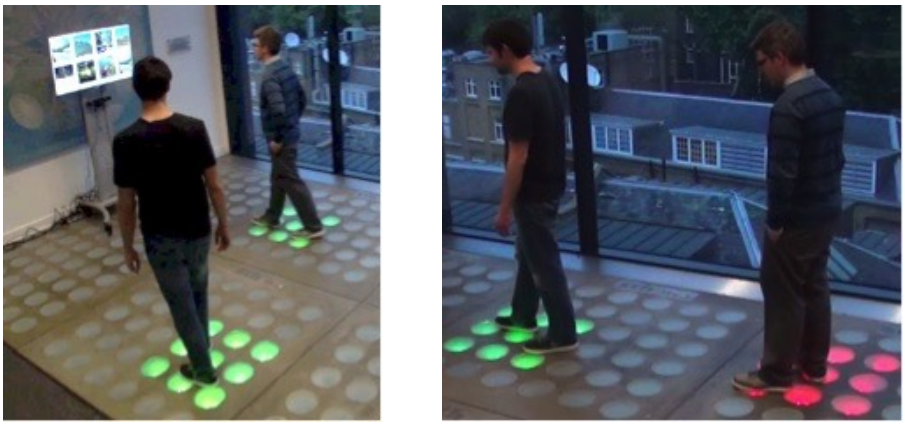
Although this is a crude mapping of tracking accuracy to different colours, we found it to be sufficient for our photo gallery application. However, other applications might require different levels of granularity. For example, for interactive proxemic game experiences (Mueller et al., 2014), tracking accuracy per body part could be helpful information for players, which could be visualized using more fine-grained halo visualization. Tracking accuracy of different body parts could be mapped to different areas of the halo; e.g., front left corresponds to left arm, back left to the left leg. Alternatively, halos could change their size depending on the area covered by the player. Of course, there is only a limited amount of information that can be conveyed using our low-resolution floor display. Revealing intricate details about the tracking quality for different body parts would require higher-resolution floor displays.

8.4.1.3 *Multi-User Halos*

Interaction around interactive surfaces is often not limited to a single person, but can involve multiple people present in the space and interacting with the display. With multiple people, information about active tracking and its fidelity becomes

even more important, because tracking problems increase with the likelihood for partial or complete occlusion.

If multiple people are present in front of the screen, each person's individual position that the system currently tracks is shown with a coloured halo (Figure 8.6a). Colour changes indicate a change in how well the user is tracked. For example, in case another person walking in interrupts the tracking camera's view of a person, the changing colour of the halo from yellow to red tells the person that they are not tracked anymore (Figure 8.6b). Similarly, if two people stand very close to one another, making it difficult for the computer vision algorithm to separate the two, the halo colour also changes to yellow.



(a) Both people tracked by the system (b) Tracking problem due to occlusion

Figure 8.6: Halos for multi-user interaction when (a) both people are visible to the system and (b) when one person is occluding the other, indicated by the red halo.

8.4.1.4 *Trails: Revealing Interaction History*

As a variation of the halo technique, the spatial trail feedback visualizes the past spatial movements of a person in the interaction area. The trails are shown as illuminated lines on the floor that light up when a person passes that particular area (Figure 8.7). The illumination fades out after a given time (in our application after 5 seconds), thus giving the impression of a comet-like trail. The colours that are used to light up the floor are identical to those of the person's halo (e.g., green, yellow, red), and therefore still provide information about the tracking quality. Because the trail visualization remains visible for a longer time, it provides information about the past movements of the people interacting with the system. Potentially, the trails could help to amplify the *honeypot effect* (Brignull and Rogers, 2003) by showing the past trails of other people moving towards the interactive display, and thus inviting other bystanders and passers-by to approach the display as well—which is why they are categorized in both phase 1 and 3 in Table 8.1.

Figure 8.7: Trails visualize the history of spatial movements of a person.

8.4.1.5 *Discussion*

Halos are a prime example of an *egocentric* visualization. They are meant to be viewed from the user's perspective, providing feedback about the tracking status. The trails variation, however, is a mostly *exocentric* technique—targeted towards being viewed from the perspective of other users—that shows information about *past* interactions. However, since the trails are still shown underneath the user's feet, and change colour depending on the user's tracking accuracy, they are simultaneously *egocentric*, and inform the user about their *present* interactions (Table 8.1a). In addition, as it potentially invites bystanders to interact with the display, the trails can serve as an invitation for *future* interactions (Table 8.1b). We can also imagine other *exocentric* halo visualizations. For example, pulsating exocentric halos could indicate open spots where users could move towards, e.g., to form teams in proxemic gaming scenarios.

8.4.2 *Phase 2. Zones and Borders: Entries and Exits for Interaction*

As mentioned earlier, people often have difficulties knowing when and how they can interact with a large public display (Jurmu et al., 2013; Müller et al., 2010). To mitigate this problem and to make it easier to discover interaction possibilities, we explicitly visualize the spatial zones for interaction and the borders of the interaction space (Figure 8.8).

8.4.2.1 *Opting-in: Proxemic Interaction Zones*

Many recent designs of large interactive displays use spatial zones around the display to allow different kinds of interaction (similar to Vogel and Balakrishnan, 2004) or change the displayed content depending on which zone a person is currently in. These zones, however, are not always immediately understandable or perceivable by a person interacting with the display. Our floor-visualizations explicitly reveal zones of interaction, allowing a person to (a) see where interaction is possible, and

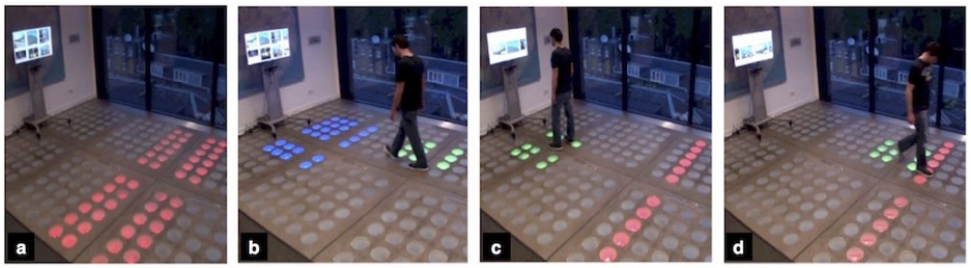


Figure 8.8: The interaction areas in front of the display represented as (a) red and (b) blue rectangular zones; (c) borders indicate thresholds to cross for (d) leaving the interaction space in front of the display.

(b) make deliberate decisions about opt-in for an interaction with the display by entering any of the zones.

We demonstrate the use of zone visualizations with the Proxemic Flow photo gallery application. Similar to earlier examples of proxemic-aware displays (Ballendat et al., 2010; Greenberg et al., 2011; Vogel and Balakrishnan, 2004), our application has discrete spatial zones around the display that are mapped to the interactive behaviour of the application on the large display. When no person is currently interacting with the system, a large red rectangular zone indicates the area furthest away from the display that triggers the initial interaction with the display (Figure 8.8a). This can be considered as the entry zone for interaction, or an area to opt-in for interaction with the system. In our current implementation, we use a 3s pulsating luminosity animation, fading the colour in and out, in our approach of balancing the goal of attracting attention while not being too intrusive. While a static colour would be possible, identifying it as part of an interactive system is potentially more difficult. Once a person enters this zone, the large display recognizes the presence of the person and tracks the person's movement—and the person's halo is shown. The first zone now disappears and a second zone—an area to interact with with the display when in front of it—appears (visible as the blue rectangle in Figure 8.8b). As shown earlier in Figure 8.4, when the person begins approaching the display, the content gradually reveals more of the photo collection on the display. The closer the person gets, the more images become revealed, a behaviour that is identical to the Proxemic Media Player (Ballendat et al., 2010). Once having entered the second zone, the person can use hand gestures in front of the display to more precisely navigate the temporally ordered photo gallery (e.g., grabbing photos, sliding left or right to move forward or back in time). Again, once the person entered that close-interaction zone in front of the display, the visualization disappears.

8.4.2.2 *Opting-Out and Exit Interaction: Borders*

While we envision zones primarily as explicit visualizations of the zones to interact, and allowing a person to deliberately engage and 'opt-in' for an interaction with the system, we can also consider visualizations that help a person leaving the in-

teraction area (i.e., opting out). We illustrate this concept with borders shown in the Proxemic Flow application. In continuation of the application example from before, once the person entered the interaction zone (blue) directly in front of the display and interacts with the display content through explicit gestures, a red border around the actively tracked interaction area around the display is shown to make the boundaries of that interaction space explicit and visible (Figure 8.8c). While we decided to dynamically show the border only in situations when a person engaged with the system, alternatively it could remain a static feature of the visualizations shown on the floor. A reason for showing a static view of the interaction boundaries with borders could be to always clearly indicate where a person can both enter but also leave the interaction area (Figure 8.8d).

8.4.2.3 *Using Zones and Borders with Multiple Users*

We can consider alternative design aspects when using zones and border visualizations with multiple users. For example, we can consider whether area visualizations are only shown to the first user entering the space and disappear once that person entered the zone, or whether the visualizations remain persistent. Showing visualizations for the first person entering a space seems most critical, and hiding the zone visualizations after the person enters a particular zone has the advantage of a floor that is less visually cluttered and therefore can help emphasizing certain parts of the visualizations (for example, make the halos stand out).

8.4.2.4 *Discussion*

In contrast to halos and trails, zones and borders are *static* visualizations. They are fixed at a certain position, and although they might only be shown at certain times, they do not follow the user. Zones and borders are also mostly *exocentric*, as they are intended to be observed from an external point of view. Nevertheless, zones can of course also be used from an egocentric perspective, when the user is inside the interaction zone (Table 8.1c). Finally, they convey cues relevant to the user's current interactions (*present*), such as borders around the actively tracked interaction area. However, zones and borders can also provide cues for future interactions, such as possible next areas to move to, or where to go to opt-out of the interaction (Table 8.1d–e).

8.4.3 *Phase 3. Waves and Footsteps: Inviting for Approach, Spatial Movement or Next Interaction Steps*

The last set of floor visualization strategies we introduce is designed to invite for approach, encourage a person's movement to a new location, and suggest the user possible next interaction steps. In particular, in this category of visualizations we introduce two strategies: waves and footsteps.

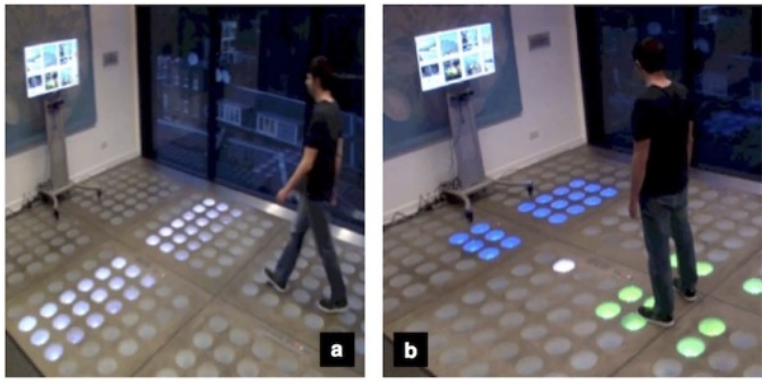


Figure 8.9: Waves inviting for interaction (a) and footsteps suggesting action possibilities (b).

8.4.3.1 *Waves: Encouraging Approach*

Our first strategy is intended for inviting people to move closer to the large display for interaction. Several strategies for encouraging approach of people have been proposed in the past (e.g., Cheung and Scott, 2013), including showing text labels, animations, graphic icon representations or using sound. With our waves technique, we leverage the output capabilities of the illuminated floor for showing looped animations of lights fading in and out, with the effect of a wave of light going towards the large screen (Figure 8.9a). Alternatively, different visual designs of the wave effect are possible, for example a circular wave effect with the large display at the centre, starting with circles having a large radius and continuously decreasing circle radius.

8.4.3.2 *Footsteps: Suggesting Next Action Possibilities*

The footsteps visualization is designed to offer a person clues about possible next interaction steps, in particular for encouraging spatial movements in the environment. The visualization shows animated footsteps—in our case these are represented through glowing circles—beginning at one location on the floor and leading to another location. This technique is inspired by earlier work of the *Follow-the-light* design (Rogers et al., 2010) that uses animated patterns of lights embedded in a carpet to encourage different movement behaviours by luring people away from an elevator towards the stairs.

To illustrate this technique, we again revisit our Proxemic Flow example application with the large display photo gallery viewer. When a person entered the interactive (i.e., tracked) space in front of the display and stands still for over 5 seconds, the floor begins the footstep animation (Figure 8.9b) to invite the person to move closer to the display. In particular, the person is invited to move to the interaction zone in front of the display where they can use mid-air gestures to further explore the image collection. The footstep animation begins directly in front of the person and leads towards the blue rectangular area highlighted in front of the display

(Figure 8.9b). The footsteps visualization strategy can be used to reveal interaction possibilities—specifically those involving spatial movements of the person. We can see this strategy being used in other contexts too for guiding or directing a user in the environment, and for encouraging movements in space, e.g., to distribute people in front of a display (Beyer et al., 2014).

8.4.3.3 *Discussion*

The visualization strategies for phase three provide cues that invite users to *future* interactions. The waves strategy is *exocentric*, as it invites bystanders to interact with the primary display. The waves pattern could be shown across the full floor display or be centralized around the primary display. Steps, on the other hand, are *egocentric* visualizations that start from underneath the person's feet, and guide them towards a certain position.

8.4.4 *Summary*

To conclude, in this section we discussed and demonstrated a set of in-situ floor visualizations that provide peripheral tracking and fidelity information with personal halos, make interaction zones and borders explicit for easy opt-in and opt-out, and provide cues inviting for spatial movement or possible next interaction steps through wave, trail, and footstep animations. This set of floor visualization strategies targets important interaction issues with large interactive surfaces that were identified in earlier research (see Section 8.2). We believe that these strategies have the potential of being applied in many different contexts, such as for games, advertisements, and other person-aware interactive systems. During informal observations of people interacting with our floor display, we noticed that essential concepts such as halos and zones were easy to understand. Future research and studies are necessary, however, to confirm these early observations. The strategies we presented here are a starting point for a collection of building blocks for how to provide in-situ visual feedback on the floor to mediate spatial interactions. In the next section, we present the Proxemic Flow software architecture and explain how we implemented the floor visualizations.

8.5 PROXEMIC FLOW ARCHITECTURE

Our aim with the Proxemic Flow software architecture was to support (1) *rapid prototyping* to allow for easy exploration of different alternative visualization strategies (e.g., alternative halo visualizations) and (2) *re-use* of existing floor visualizations for different applications and settings (i.e., integration of zones or halos with just a few lines of code).

Proxemic Flow consists of four main technical components: (i) the hardware setup of the illuminated floor, (ii) the user tracker and (iii) floor renderer that are connected together using (iv) the floor toolkit, a modular API written in the C# programming language. The existing hardware setup of the floor was developed by

Bird et al. (2013) in the foyer of the University College London's computer science department. I implemented the user tracker, floor renderer and floor toolkit and integrated it with the hardware setup by Bird et al.. The user tracker is responsible for tracking users in the space in front of the display, and for mapping these positions to positions on the floor. The floor renderer is a C# component that processes updates to a *floor scene* and renders these to an offline *floor bitmap*. Every render cycle, render updates are sent over the network to a Processing sketch, which interprets these and sends update messages over a serial connection to the Arduino microcontroller that steers the different light units. The floor toolkit provides a number of visual rendering primitives (static and dynamic visualizations) to build up the floor scene. In addition, it allows floor applications to respond to user tracking events (e.g., when a new user is detected or a user's position has changed). We will now explain these different components in more detail.

8.5.1 *Hardware Setup of the Interactive Floor Display*

The floor that we use for our setup consists of 288 light wells set in concrete, of which 216 wells are fitted with a custom LED light unit, as shown in Figure 8.10. As described by Bird et al. (2013), each custom light unit consists of four RGB LEDs cut from an LPD8806 LED strip, joined together and mounted onto a plastic cap which fits into the concrete surface from the floor below. The light units are connected in series and powered by three modified ATX power supplies. An Arduino Mega controls the floor display. Each of the light units can be set to one of around 2 million colours and the whole array can be updated at a rate of up to 25 fps. This effectively turns the floor into a large display with a resolution of 12x18 pixels.

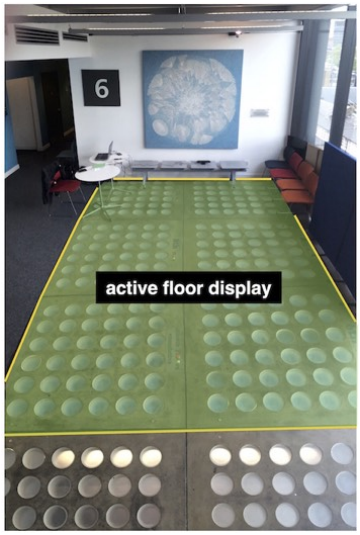


Figure 8.10: The floor displays consists of the 216 light wells as indicated by the shaded area.

8.5.2 *Tracking Users*

Users are currently tracked across the floor using a single Microsoft Kinect depth camera (version 1.0) and the Microsoft Kinect SDK, which allows us to track up to six simultaneous users (with skeleton data available for two users).

As the position of a user can be represented in the 2D (x, z) plane (the user's vertical position is ignored), a simple affine matrix transformation suffices to map the (x, z) coordinates as given by the Kinect camera to a (i, j) position in the floor grid. To set up the system, a four-point calibration is performed to map positions seen by the Kinect to the corresponding floor positions, after which the corresponding transformation matrix is calculated. Calibration settings can be saved and reused for future sessions.

We currently represent the tracking accuracy for a specific user—as used for determining the colour of the personal halos (Section 8.4.1)—as a value in the range $[0, 1]$. For the Kinect sensor, this value is obtained by calculating an arithmetic average over the accuracy of the skeleton joints, which have one of three states: *tracked*, *inferred*, or *not tracked*. We currently assign the value 1.0 to *tracked* joints, 0.3 to *inferred* joints and 0.0 to joints that are *not tracked*. Green halos are shown for average accuracies over 0.7; halos turn red when the accuracy drops below 0.3; and yellow halos are shown for accuracies between 0.3 and 0.7. These specific tresholds have been selected based on empirical observations, but can be easily changed.

Every time an update is received from the Kinect skeleton stream, we create a *TrackingUpdate* object that encapsulates all necessary information for the floor, such as the user's position on the floor grid and the tracking accuracy. Floor applications can subscribe to these events and respond accordingly, for example by showing halos indicating the current position and tracking accuracy of all users, or by updating the floor display when users cross borders or enter zones.

8.5.3 *Rendering Pipeline: Updating the Floor Display*

The *rendering pipeline* is the core part of the architecture and handles all updates to the floor display (Figure 8.11).

The *floor scene* (Figure 8.11a) is the logical representation of the graphics on the floor and consists of an ordered set of layers (e.g., halos, zones, borders, inviting animations) that are rendered on top of each other, each of which can be enabled or disabled. There is a logical priority of layers: low level primitives (Figure 8.11a_1) such as zones and borders are rendered first, followed by dynamic content such as the steps or wave animations, and finally the personal halos which are always visible, even when they collide with other visuals. Animations consist of a set of frames (Figure 8.11a_2), which are rendered in each update (one after the other). Animations can be looped, in which case they return to the first frame after the last frame has been rendered.

The *floor renderer* (Figure 8.11b) uses a timer to allow sending update messages to the floor at a fixed rate. A floor update message is represented by a *FloorBitmap* object (Figure 8.11c), which is an 18×12 grid of colour values for each of the light

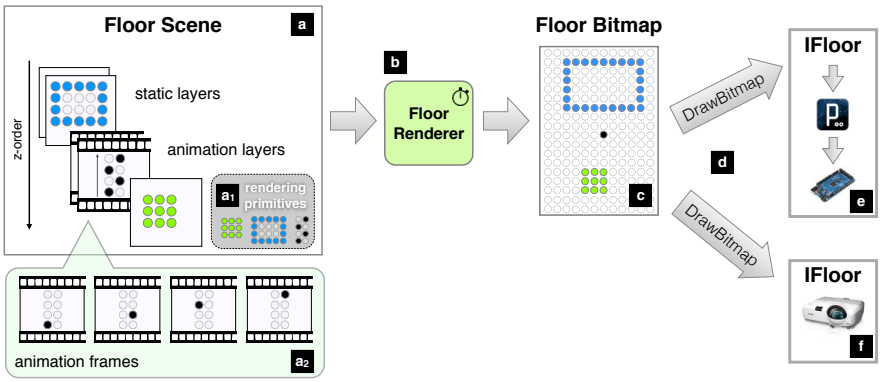


Figure 8.11: The Proxemic Flow rendering pipeline. Visualizations on the floor display are abstracted in a *floor scene* (a). This floor scene is processed by the *floor renderer* (b), resulting in (c) a *floor bitmap* (an abstraction of a floor display update) that is send over the network to the connected floor displays that implement the *IFloor* interface (d). We also implemented a projected floor display (f).

wells in the grid. Every tick, the rendering pipeline sends a floor update message to the connected *IFloor* instance by calling its *DrawBitmap* methodb (Figure 8.11d). The default *IFloor* instance forwards messages to the Processing sketch, which is connected to the *Arduino* microcontroller that controls the light units in the LED floor display (Figure 8.11e).

8.5.4 *Proxemic Flow Toolkit*

The Proxemic Flow toolkit is event-driven and abstracts from the underlying floor display and tracking solutions to allow for rapid exploration of visualization strategies. It provides a set of reusable visualizations (e.g., halos, zones, guiding animations) to easily modify the visualizations for different settings and applications. These rendering primitives are connected to tracking updates in floor applications. All graphics and animations implement *IRenderPrimitive* and *IAnimationPrimitive* interfaces respectively, both of which can be translated into a floor bitmap. Application developers may add new visuals by simply providing new classes that implement these interfaces. This allows the *rendering pipeline* to be agnostic to the specifics of the graphics being shown on the floor. All that is required is that graphics and animations can be transformed into a floor bitmap.

For example, Listing 8.1 shows the code that is necessary to implement a *pulsating dot* animation that cycles through a list of colours. It forms the basis for implementing the trails visualization, which essentially continuously renders slowly pulsating dots at the user's current position. The toolkit also makes it easy to respond to tracking updates and perform basic operations such as checking whether the user is within a certain zone. Listing 8.2 shows an excerpt that renders a *steps* animation

towards a specific interaction zone if the user is not in that zone and has not moved for 5 seconds (as indicated by the *dwell time*).

Listing 8.1: Code to implement a pulsating dot animation at a specific location.

```
class PulsatingDot : AbstractAnimationPrimitive
{
    private FloorPosition position;
    private List<FloorColor> colors;

    public PulsatingDot(FloorPosition position, List<FloorColor> colors)
    {
        this.position = position;
        this.colors = colors;

        // Create frames
        foreach (FloorColor color in this.colors)
        {
            RenderLayer layer = new RenderLayer();
            layer.Add(new Dot(this.position, color));
            this.animation.AddFrame(layer);
        }
    }
}
```

Listing 8.2: Code that shows a steps animation towards 'frontZone', if the user is currently not in that zone and has been standing still for more than 5 seconds.

```
public override void OnTrackingUpdate(List<TrackingUpdate> updates)
{
    base.OnTrackingUpdate(updates);
    foreach (TrackingUpdate u in updates)
    {
        if (u.Tracked)
        {
            if (!stepsShownAlready && !this.frontZone.Contains(u.Position) &&
                u.DwellTime > 5000)
            {
                // Steps animation
                StepsAnimation steps = new StepsAnimation(
                    u.Position,
                    this.frontZone.LowerRight.Row,
                    FloorColor.White);
               stepsLayer.AddAnimation(steps);
            }
        }
    }
}
```

8.5.5 *Generalizability*

The Proxemic Flow architecture and toolkit enabled us to rapidly prototype different floor visualizations for mediating surface interactions. We believe our floor visualizations are a starting point for exploring alternative strategies, applying them in different application contexts, and evaluating their use in practice—and in particular with in-the-wild deployments (Rogers, 2011) and testing of floor visualizations supporting interactions in public settings.

In future work, we would like to extend Proxemic Flow further beyond the specifics of our current floor setup. Ideally, the floor visualizations should be supported on different floor displays (e.g., using projectors or FTIR as described by Bränzel et al., 2013), and also be able to use a variety of other tracking solutions (e.g., 2D cameras with markers, other depth cameras, or optical trackers such as VICON[4]). Here, we briefly explain how our current architecture was already built with this flexibility in mind, and what changes need to be made to be able to extend it further.

8.5.5.1 *Alternative Floor Display Implementations*

Figure 8.12 shows an alternative rendering solution we implemented in order to show visuals on arbitrary surfaces, based on an overhead projector mounted to the ceiling. It uses the same floor toolkit as described earlier, so that applications written once run without modification. We have successfully verified this by first implementing visuals using the projector setup in the lab at Hasselt University, which were afterwards tested on the LED floor display at University College London where they ran without changes.

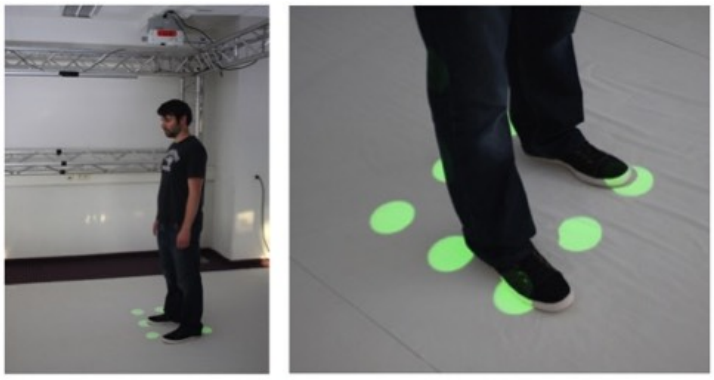


Figure 8.12: Alternative floor display using a ceiling-mounted short-throw projector.

For the projector-based implementation, a separate Windows Presentation Foundation (WPF)[5] window that renders the floor grid that is projected onto the floor.

4 http://www.vicon.com/
5 http://www.windowsclient.net/

As mentioned before, the toolkit's rendering pipeline sends updates to connected objects that implement the *IFloor* interface. All that is needed for an alternative floor display is changing this object to another *IFloor* instance. Our projection-based floor responds to floor update messages by changing an internal model of the floor grid, which is then also updated in the WPF view.

Although this approach makes it easy to run applications developed with the projector setup unmodified on the 18x12 LED floor (and vice-versa), supporting higher-resolution floor displays would require some additional changes. Especially the floor scene model and the floor renderer would be impacted. Our toolkit assumes that all rendering primitives can be transformed into an off-screen floor bitmap (which is currently 18x12 pixels). However, a minor modification to support higher-resolution displays would be to support floor bitmaps of arbitrary sizes (e.g., 1024x768). Rendering primitives should then be able to render to different floor bitmap formats, depending on which IFloor instance it is rendering for. To render high-resolution floor bitmaps using a projector, they could be converted into offline bitmaps (images), that are then rendered as a widget inside the WPF window and projected onto the floor. Although we have not verified this, we believe it would not severely impact performance for a projector-based setup[6]. Although our current architecture does not explicitly support vector graphics, vector rendering primitives can be used as long as they are transformed into a bitmap in the final step.

8.5.5.2 *Alternative User Tracking Implementations*

Our toolkit is reasonably decoupled from the specifics of the tracking technology we use. The *TrackingUpdate* object abstracts from the underlying tracking hardware; the same information could be produced by another tracking solution (e.g., a VICON motion tracker) and passed to the rendering pipeline. Although this is a fairly simplistic approach, a more sophisticated mechanism using the decorator pattern (Gamma et al., 1995) could be integrated in order to allow each tracking technology to specify what kind of data it provides (e.g., as in Marquardt et al., 2011).

8.6 DISCUSSION

In this section, we briefly discuss a number of design considerations we feel are important for peripheral floor visualizations. We elaborate on *what* visualizations can be used, *when* these visualizations are revealed to users, and finally *where* they are shown.

8.6.1 *What to Show?*

Our approach is minimalistic on purpose: we reduced the visualizations to essential *cues* that are easy to read and require minimal visual bandwidth. These can be fur-

6 A 1024x768 LED floor display, on the other hand, will probably need to be engineered in a different way to achieve similar performance.

ther enriched (e.g., finer grained spatial movement cues), but it is important to avoid a visually cluttered floor with—perhaps even animated—visualizations that distract the user. As mentioned earlier in Section 8.3, it is not our intention with Proxemic Flow to build an attention-grabbing floor display with the goal of maximizing the number of people interacting with the primary display. The visualizations should be there when needed, but not unnecessarily draw the user's attention. Moreover, the floor visualizations should not detract from interacting with the primary display, as the floor plays a secondary, assisting role.

Nevertheless, at times it can be necessary to move to the foreground and inform the user about what is happening (similar to Ju et al.'s *system demonstration*). One possible future direction is to combine the floor visualizations with other modalities to improve the users' awareness. While audio feedback tends to be inappropriate in public spaces, vibrotactile feedback on the floor (Visell et al., 2009) would allow users to 'feel' when they are crossing a border or entering a zone without having to direct their gaze to the floor. Another interesting direction for future work is to investigate how users cannot only be provided with information about action possibilities, but also information about the purpose of those actions (i.e., feedforward).

8.6.2 *When to Show Information?*

An important design consideration is balancing the timing when information needs to be shown to the user. Several of our introduced visualizations appear ahead of time to inform users (e.g., triggering zones, borders) while others appear only when the users are in the tracking area (e.g., halos). Trails remain visible for a certain period to provide historic action traces to other users (see Table 8.1 on page 155). This difference in information needs depends on the situation. In summary, we believe the following aspects contribute to when information needs to be shown:

- *Single versus multi-user interaction*: it can be useful to provide visualizations that provide information to *others*, either for people who are interacting with the display at another location, or for bystanders or passers-by who might be able to use the floor visualizations to approach the display as well. This corresponds to the egocentric versus exocentric perspective in Table 8.1. As mentioned before, we can imagine that the trails visualization could amplify the *honeypot effect* (Brignull and Rogers, 2003) even more.

- *Avoid clutter*: information should appear only for a limited time, since it might otherwise result in a cluttered floor display. We believe there is also a lot of value in avoiding clutter by relying on 'just-in-time' visualizations (Wigdor and Wixon, 2011) that are only shown when the user has indicated interest in performing a certain action.

- *Application type*: applications that aim for a high throughput of users (e.g., information displays in crowded places), might want to employ snappier floor visualizations, stimulating people to only use the system for a limited time and then move on. We can imagine our floor visualizations to be used for

audience shaping as well (e.g., similar to techniques used by Beyer et al., 2014). On the other hand, applications that aim to involve the user for a longer period of time, might want to relax these time constraints and allow users to gain a deeper understanding of the system.

8.6.3 *Where to Show Information?*

It is important to think about where visualizations are shown, as users might not always be paying attention (e.g., they might not look at the floor much when they are directly in front of the display), or might be unaware of certain visualizations because they are outside their field of view (e.g., imagine a zone that appears behind the user). While inviting and guiding strategies can direct the user's attention, care should be taken to not design floor visualizations that require users to always pay attention to the floor. An interesting opportunity for future work is to investigate how users' peripheral view—which is very sensitive to motion (Heun et al., 2012)—can be used (sparingly) to draw their attention.

One could argue that users might not notice halos, since these are visualized directly underneath their feet. However, as mentioned before, during initial observations, we noticed that people became aware that the floor was actually a display as they entered the tracking area (e.g., by first noticing the border visualizations, and then their personal halo). A quick glance at the visualizations should suffice. Moreover, they are only meant to be used when users are unsure about interacting in the space in front of the display. Especially the egocentric visualizations should be easy to locate. Finally, for multi-user scenarios, it can be useful to design floor visualizations that make it easier to learn how to interact with the display by observing others, a concept known as *external legibility* (Zigelbaum, 2008). Earlier studies have indeed found that very visible ways of interacting with displays (e.g., mid-air gestures) increase the honeypot effect (Müller et al., 2012, 2014) and result in users copying each other's behaviour (Walter et al., 2013).

8.7 CONCLUSION

We presented dynamic in-situ floor visualizations for revealing and mediating large surface interactions. With Proxemic Flow, we demonstrated three categories of visualizations: (1) personal halos and trails that provide peripheral information about current tracking and tracking fidelity; (2) interaction zones and borders for easy opt-in and opt-out; and (3) wave and footstep cues that invite users for movement across the space or possible next interaction steps. We believe that leveraging the floor as a peripheral, secondary output device for showing such in-situ feedback can help to mediate interactions with large surfaces. Our proposed floor visualization strategies aim target several important interaction problems with large interactive surfaces that were identified in earlier work (see Section 8.2). We feel these three categories of in-situ floor visualization strategies have the potential to improve walk-

up-and-use interaction with future large surface applications in different contexts, such as gaming, or for entertainment or advertisement purposes.

During informal observations of people interacting with our floor display, we noticed that essential concepts such as halos and zones were easy to understand. We believe this is due to their visual simplicity and the fact that users only need to pay attention to the floor occasionally—which would not be the case for more complex visuals or text. Further studies and long-term deployments, however, are necessary to confirm our early observations. In future work, we would also like to improve the extensibility of our architecture to support a wider variety of different floor displays, including ones that support higher resolutions.

9 CONCLUSIONS

This dissertation has shown how to design for intelligibility and control in ubiquitous computing applications—and context-aware systems in particular. This final chapter recapitulates the contributions made by the presented design principles and techniques, and concludes with an outlook on future work.

9.1 RESTATEMENT OF CONTRIBUTIONS

In this dissertation, I have conducted a design space exploration to inform the design of future ubicomp systems that provide support for intelligibility and control. I introduced (1) a *design space* that can guide designers in exploring various ways to support intelligibility and control, presented (2) *general design principles and techniques* that can be applied in a wide range of ubicomp scenarios, and described (3) an *in-depth case study* of supporting intelligibility and control in proxemic interactions that can serve as inspiration for designers looking to support intelligibility and control in different ubicomp scenarios.

In particular, I made the following major contributions:

1. A *design space for intelligibility and control* (Chapter 3) that captures different decisions that designers face when adding support for intelligibility and control. This design space can be used as an analytical tool and can help designers explore alternative designs.

2. An *in-depth exploration of the timing dimension* in the previous design space. In particular, I contribute three general techniques that can be used at three different times during the interaction: before, during, and after actions.

    a) *Before*—The design principle *feedforward* (Chapter 5), which informs users of the results of their actions. I provide a new definition of feedforward, further disambiguate it from feedback and affordances. In addition to discuss several existing examples of feedforward, I describe the *Feedforward Torch* technique. Finally, I identify four new classes of feedforward: *hidden*, *false*, *sequential*, and *nested* feedforward.

    b) *During*—The design principle *slow-motion feedback* (Chapter 6), which is aimed at allowing users to intervene during system actions by having the system slow down when taking action and provide intermediate feedback. I illustrate the application of slow-motion feedback in our *Visible Computer* system to provide real-time, in-place feedback in context-aware environments using projected visualizations. Furthermore, I introduce a design space to reason about when and how feedback is provided, and use it to analyse notable existing applications of slow-motion feedback.

c) *After*—The ability to *pose why questions about the behaviour of a context-aware system* (Chapter 7). This allows users to gain an understanding of how the system works by receiving intelligible explanations of why it acted in a certain way. I introduce *PervasiveCrystal*, a framework for building context-aware applications that can provide answers to 'why?' and 'why not?' questions about their behaviour.

3. A *case study* (Chapter 8) in supporting intelligibility and control for *proxemic interactions*, a subdomain of context-aware computing. I discuss the design and implementation of dynamic peripheral floor visualizations to address interaction challenges with proxemic-aware interactive surfaces. The *Proxemic Flow* system uses a floor display that plays a secondary, assisting role to aid users in interacting with the primary display. The floor informs users about the tracking status, indicates action possibilities, and invites and guides users throughout their interaction with the primary display.

9.2 FUTURE WORK

Important limitations and possible extensions were discussed in each preceding chapter. This chapter briefly discusses some larger future research directions.

9.2.1 *Intelligibility and Control "In the Wild"*

I conducted initial informal evaluations of the proposed techniques, which are described in Sections 5.6.4 (Feedforward Torch, pg. 100), 6.3.5 (The Visible Computer, pg. 120), and 7.6 (why questions, pg. 141). However, to fully understand the benefits of different techniques, we believe an important direction for future work is to study intelligibility and control "*in the wild*" (Rogers, 2011). In-the-wild studies could help us gain detailed insights about the impact of intelligibility and control with respect to real-world concerns in different settings, and could also reveal how intelligibility and control features are used over time (e.g., whether or not there is a novelty effect).

There have been several studies on the effectiveness of intelligibility and control techniques, such as the work by Lim and Dey, which contributed valuable insights into the effectiveness of different types of explanations (e.g., Lim et al., 2009; Lim and Dey, 2009, 2011b,a). Most of these studies, however, were conducted in lab settings, simulated using questionnaires, or have only been deployed over a short period of time. Rogers (2011) stresses that human-computer interaction in the real world is more *messy* than in the lab, as people tend to be much more unpredictable. As a first step, researchers have started to explore intelligibility concerns in real-world products. A recent example is the work by Yang and Newman (2013), that studied users' experiences with the Nest smart thermostat in several households. An interesting observation was that users had little interest in learning about the thermostat's workings as an independent activity. In the 'messiness' of their daily lives, home-owners had little motivation to go through the effort of understanding

the thermostat's behaviour. Yang and Newman (2013) therefore suggest to provide what they call *incidental intelligibility*, which is integrated into the tasks users are trying to accomplish, which corresponds to *embedded* techniques in our design space (see also Section 3.3.3).

We took a similar approach of 'just-in-time' intelligibility with our floor visualizations in the Proxemic Flow system (Chapter 8). This work was also informed by several interaction challenges that were observed in earlier in-the-wild studies of large interactive surfaces in semi-public settings (e.g., Ojala et al., 2012; Müller et al., 2009b, 2012). Due to its characteristics as a multi-user context-aware application that relies on implicit interaction and needs to support walk-up-and-use interaction, our Proxemic Flow setup provides interesting opportunities for studying intelligibility and control in real-world settings. Furthermore, additional floor visualizations and techniques such as feedforward could be integrated, compared and studied. An "in the wild" study would also open up opportunities to investigate multi-user intelligibility and control in a natural setting, leading to the next opportunity for future work.

9.2.2 *Multi-User Intelligibility*

Most existing research on intelligibility, the work in this dissertation included, primarily targets intelligibility for single-user scenarios. Future research should broaden the scope to investigate intelligibility in social settings and for multi-user ubicomp applications. There are several issues that need to be addressed when providing intelligibility in ubicomp scenarios involving multiple users. First, for multi-user systems that act autonomously based on the sensed context, it will also be necessary to provide awareness about the actions of other users and of who the system is responding to, which is also known as *accountability* (Bellotti and Edwards, 2001). Bellotti and Edwards argue that, next to intelligibility, accountability is an important principle to support in multi-user context-aware systems that seek to mediate user actions that impact others. In addition to the need for awareness of the actions of others and accountability of one's actions, *privacy* (Bellotti and Sellen, 1993) is an important open issue for multi-user intelligibility. As a consequence of being intelligible, systems might reveal sensitive personal information. Care should be taken to ensure that other users do not have access to that information, both for remote users and nearby users (e.g., when designing intelligibility for co-located multi-user ubicomp environments).

In particular, in the context of large interactive surfaces, there are a few future research avenues worth exploring to address privacy challenges with intelligibility. First of all, researchers could build on existing work on using proxemics to control privacy by when showing sensitive information to the user. For example, Vogel and Balakrishnan (2004) relied on the user's body being able to shield information from bystanders. Their display revealed increasingly detailed and personal information when moving closer, and created a split-screen setup when other users approached. Similarly, intelligibility could be provided on the user's personal device, to maintain the user's privacy and avoid disturbing other users in their activities (e.g., Cheung

et al., 2014). Another interesting opportunity is to provide awareness of possible privacy intrusions, as explored by Brudy et al. (2014) by tracking other people in the area surrounding a public display. In fact, this particular type of awareness can be seen as an example of *accountability*, in which the system amplifies users' shoulder surfing behaviour and makes them accountable for their actions.

In Chapter 8, we briefly touched upon multi-user intelligibility. When intelligibility does not reveal sensitive information, I feel that it can be useful to assist users in working around sensing limitations together. For example, by providing tracking feedback with our personal halo visualizations, users are able to regulate their own behaviour and move aside when they notice that they are occluding another user (Figure 8.6b, pg. 158). An interesting possibility for future work is to build upon this and further investigate how we can leverage floor visualizations to make users explicitly aware of how they are impacting each others experiences.

9.2.3 *Further Exploring and Extending the Design Space*

A clear direction for future research is to further explore the design space for intelligibility and control (Chapter 3, pg. 36). Figure 9.1 situates the presented prototypes and case study in the design space for intelligibility and control (see also Table 3.7 on page 48). The prototypes correspond to different alternatives for the timing dimension, and also explore several other dimensions in the design space. In the Proxemic Flow case study, we provide an example of a domain-specific technique that spans across the timing dimension. Figure 9.2 illustrates the main sub-areas of the design space that were explored in this dissertation. The lighter areas in Figure 9.2 reveal some limitations and additional areas of exploration I have not investigated in-depth. Based on this overview, I provide a few suggestions for future work.

9.2.3.1 *Providing Intelligibility Using Other Modalities*

As is apparent from Figure 9.2, the research in this dissertation mostly relies on providing intelligibility using visual information. The visual channel has a number of important advantages, such as being able to easily convey detailed information. It will therefore likely remain the dominant modality that designers will target to support intelligibility. Nevertheless, there are some disadvantages of visual information, such as the need for users' visual attention or the problem of information overload. Consequently, there are interesting opportunities for future research to combine multiple modalities. In particular, other modalities could draw users' attention, or provide necessary cues when users' visual attention is needed elsewhere. For example, as explained in Chapter 8, borders and zones could be revealed using additional haptic feedback integrated in the floor surface. To further explore multimodal user interfaces for intelligibility and control, the design space can be refined in terms of existing design spaces for multimodal interaction. For example, the modality dimension could be refined in terms of the CARE properties (Coutaz et al., 1995) to capture how different modalities are combined. Finally, future research could also investigate how the techniques explored in this dissertation could be mapped to

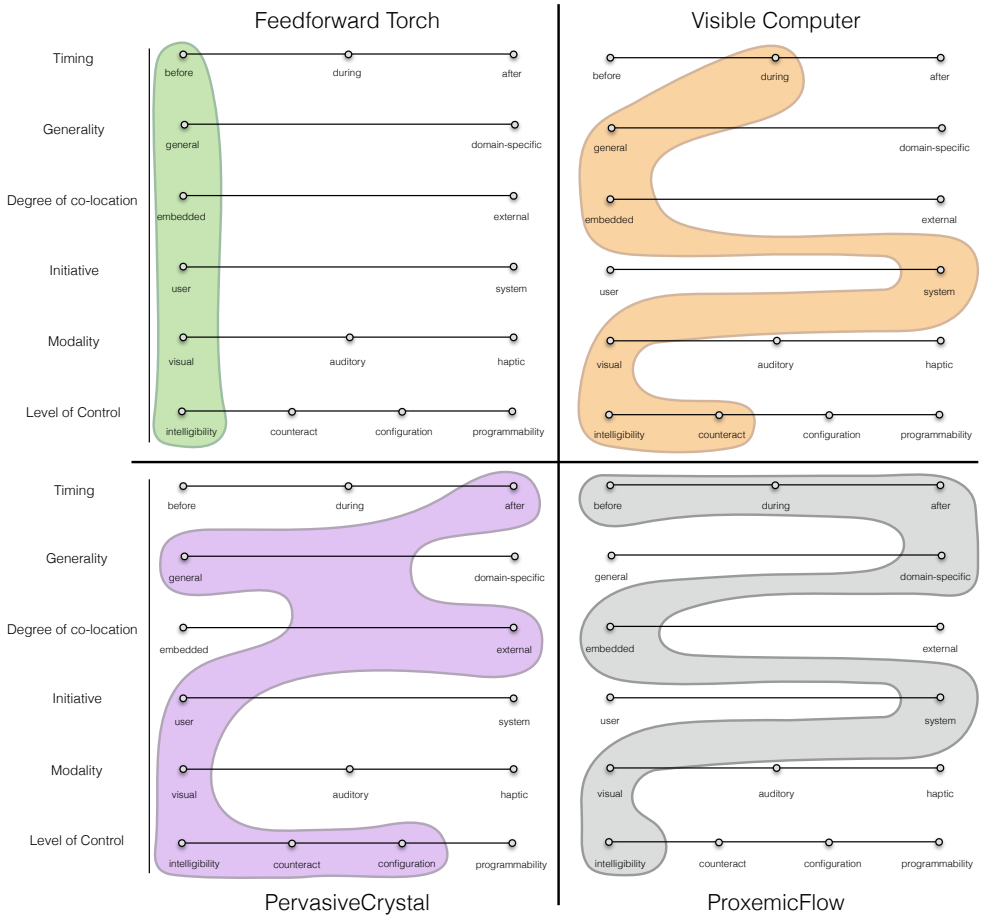


Figure 9.1: The different prototypes and case study situated in the design space for intelligibility and control.

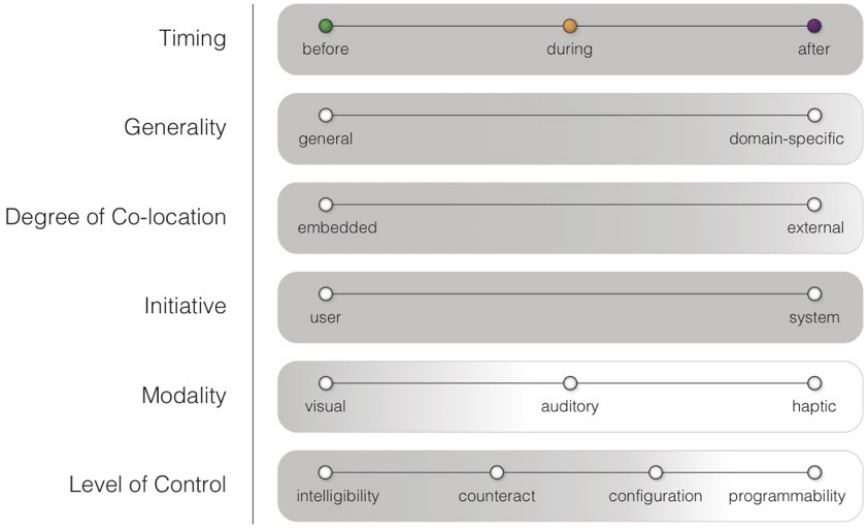


Figure 9.2: The combination of dark areas shows the subspace within the design space that was covered in this dissertation, while lighter areas indicate additional open areas for exploration.

other modalities (e.g., slow-motion feedback could rely on haptic or audio cues that slowly increase in intensity or speed).

9.2.3.2 *End-User Programmability of Ubicomp Environments*

This dissertation primarily focuses on providing users with *opportunities for control*. An active area of research explores how non-technical end-users can define and reprogram the behaviour of their ubicomp environments (e.g., Rodden et al., 2004; Dey et al., 2006; Kawsar et al., 2008). This challenge is gaining importance with the increasing availability of Do-It-Yourself (DIY) smart home technologies (Mennicken et al., 2014; Coutaz et al., 2014).

There are a couple of possible extensions to my work that might increase the *level of control* (Figure 9.2). For example, PervasiveCrystal (see Chapter 7) could be extended to not only explain how different context rules influence the environment's behaviour, but also allow users to modify and reprogram these rules. As a basic first step, PervasiveCrystal could learn from users' interactions with the system, and suggest modification of certain rules. For example, when the user always reverts the effect of a particular rule, PervasiveCrystal could suggest to change (or even remove) it.

There is potential for techniques that allow users to demonstrate what they want to achieve, an approach which has been successfully applied for automating web-based processes (Leshed et al., 2008) and in design tools for non-technical users (Hartmann et al., 2007a,b). Additionally, researchers are also exploring how to facilitate end-users to understand and modify their machine-learned systems (e.g.,

Kulesza et al., 2009, 2011), which might also provide useful insights for ubicomp systems.

9.2.3.3 *Broadening the Design Space with Additional Dimensions*

The design space for intelligibility and control is, of course, not exhaustive. More dimensions could be added to extend the design space and provide additional insights. Possible dimensions that could be relevant for intelligibility techniques, and might be interesting to explore in future research are the *level of detail* at which information is provided, and whether information is provided at *discrete intervals* or *continuously*. These dimensions were briefly discussed in Chapter 5 (see Table 5.1, pg. 79) and the level of detail was also considered in Chapter 6 (see Figure 6.3, pg. 107).

9.2.4 *Beyond Context-Aware and Ubiquitous Computing Applications*

A number of techniques that were presented in this dissertation extend beyond context-aware and ubicomp applications. An interesting area for future research is to investigate how these techniques can be applied in other domains.

For example, there is an increasing body of work on improving the *learnability* of complex, specialized desktop applications such as Adobe Photoshop or Autodesk AutoCAD (e.g., Grossman et al., 2009; Matejka et al., 2009; Lafreniere et al., 2013). Grossman et al. (2009) provide an overview of learnability problems and 'Understanding' is particularly relevant to the work presented in this dissertation: "This problem means that users were aware of a single, specific, tool or function, able to locate it, but could not figure out how to use it." Grossman et al. argue that the most promising techniques describe how to use a specific function, and additionally provide an (in-context) *demonstration*. An example of this type of technique is the integration of video toolclips into tool palettes and toolbars (Grossman and Fitzmaurice, 2010). There are opportunities for feedforward in this domain, specifically to help users understand what exactly happens when performing certain operations. One promising direction is contextualizing such previews to the user's current document, and providing an exploration mode in which users can quickly understand different commands. This shares similarities with the work by Vanacken et al. (2008) for multi-touch tabletops, in which interactive gesture demonstrations provide a live preview of how gestures affect the objects in the current application. In addition, slow-motion feedback could help to ease the transition from novice to expert in complex desktop applications. While most applications do provide feedback about background actions, slow-motion feedback could increase awareness of these actions for novice users. Example of such background actions are automatic corrections to typed words in a word processor, or automatic conversion of numbers to dates in a spreadsheet application. These actions could be performed more slowly for novice users to highlight what the system did, improving user awareness.

In addition to complex desktop applications, other research areas in HCI that share similar issues regarding understanding, visibility, discoverability and implicit

interaction could also potentially benefit from the techniques presented in this work. In particular, interaction styles that rely heavily on sensing are obvious candidates (Bellotti et al., 2002), such as tangible computing, mid-air gestural interaction and multimodal interaction.

9.3 CLOSING REMARKS

With an increasing number of domestic, mobile and wearable context-aware devices available, it is important to be able to smoothly integrate these technologies into our lives. This dissertation asserts that ubicomp environments and context-aware systems should provide support for *intelligibility* and *control* and investigates how we can *design* for these concerns. My research empowers interaction designers, researchers and developers to envision and realize a broader range of solutions to support intelligibility and control. I hope that the work presented here will inspire designers to build applications for the post-desktop computing age that are predictable, understandable, and leave the user in control.

A

LIST OF PUBLICATIONS

JOURNAL ARTICLES

- Sebastian Boring, Saul Greenberg, **Jo Vermeulen**, Jakub Dostal, and Nicolai Marquardt. The Dark Patterns of Proxemic Sensing. *Computer*, 47(8):56–60, Aug 2014. ISSN 0018-9162.

CONFERENCE PAPERS

- Sarah Mennicken, **Jo Vermeulen**, and Elaine M. Huang. From Today's Augmented Houses to Tomorrow's Smart Homes: New Directions for Home Automation Research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, 2014. ACM. ISBN 978-1-4503-2968-2.
- **Jo Vermeulen**, Kris Luyten, Karin Coninx, and Nicolai Marquardt. The Design of Slow-Motion Feedback. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 267–270, 2014. ACM. ISBN 978-1-4503-2902-6.
- Saul Greenberg, Sebastian Boring, **Jo Vermeulen**, and Jakub Dostal. Dark Patterns in Proxemic Interactions. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 523–532. 2014. ACM. ISBN 978-1-4503-2902-6.
- **Jo Vermeulen**, Kris Luyten, Elise van den Hoven, and Karin Coninx. Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1931–1940, 2013. ACM. ISBN 978-1-4503-1899-0.
- Steven Houben, Jakob Bardram, **Jo Vermeulen**, Kris Luyten, and Karin Coninx. Activity-Centric Support for Ad Hoc Knowledge Work – A Case Study of co-Activity Manager. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2263–2272, 2013. ACM. ISBN 978-1-4503-1899-0.
- **Jo Vermeulen**, Kris Luyten, and Karin Coninx. Intelligibility Required: How to Make us Look Smart Again. In *Proceedings of the 10th Romanian Conference on Human-Computer Interaction*, ROCHI '13, 2013.
- **Jo Vermeulen**, Kris Luyten, and Karin Coninx. Understanding Complex Environments with the Feedforward Torch. In *Ambient Intelligence*, volume 7683 of

Lecture Notes in Computer Science, pages 312–319. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34897-6.

- **Jo Vermeulen**, Fahim Kawsar, Adalberto L. Simeone, Gerd Kortuem, Kris Luyten, and Karin Coninx. Informing the Design of Situated Glyphs for a Care Facility. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, VLHCC '12, pages 89–96, 2012.

- Steven Houben, **Jo Vermeulen**, Kris Luyten, and Karin Coninx. co-Activity Manager: Integrating Activity-Based Collaboration into the Desktop Interface. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 398–401, 2012. ACM. ISBN 978-1-4503-1287-5.

- Fahim Kawsar, **Jo Vermeulen**, Kevin Smith, Kris Luyten, and Gerd Kortuem. Exploring the Design Space for Situated Glyphs to Support Dynamic Work Environments. In *Pervasive Computing*, volume 6696 of *Lecture Notes in Computer Science*, pages 70–78. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21725-8.

- **Jo Vermeulen**, Geert Vanderhulst, Kris Luyten, and Karin Coninx. PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications. In *Intelligent Environments (IE), 2010 Sixth International Conference on*, pages 271–276, 2010. IEEE.

- **Jo Vermeulen**, Jonathan Slenders, Kris Luyten, and Karin Coninx. I Bet You Look Good on the Wall: Making the Invisible Computer Visible. In *Ambient Intelligence*, volume 5859 of *Lecture Notes in Computer Science*, pages 196–205. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-05407-5.

- Nasim Mahmud, **Jo Vermeulen**, Kris Luyten, and Karin Coninx. The Five Commandments of Activity-aware Ubiquitous Computing Applications. In *Digital Human Modeling*, volume 5620 of *Lecture Notes in Computer Science*, pages 257–264. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02808-3.

- Jan Meskens, **Jo Vermeulen**, Kris Luyten, and Karin Coninx. Gummy for Multi-Platform User Interface Designs: Shape me, Multiply me, Fix me, Use me. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '08, pages pp. 233–240. ACM

- **Jo Vermeulen**, Yves Vandriessche, Tim Clerckx, Kris Luyten, and Karin Coninx. Service-interaction Descriptions: Augmenting Services with User Interface Models. In *Engineering Interactive System*, volume 4940 of *Lecture Notes in Computer Science*, pages 447–464. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-92697-9.

- Kris Luyten, Kristof Thys, **Jo Vermeulen**, and Karin Coninx. A Generic Approach for Multi-Device User Interface Rendering with UIML. In *Computer-Aided Design of User Interfaces V*, pages 175–182. Springer Netherlands, 2007. ISBN 978-1-4020-5819-6.

BOOK CHAPTERS

- James Helms, Kris Luyten, Robbie Schaefer, **Jo Vermeulen**, Marc Abrams, Adrien Coyette, and Jean Vanderdonckt. Human-Centered Engineering of Interactive Systems with the User Interface Markup Language. In *Human-Centered Software Engineering*, pages 139–171, 2009. Springer-Verlag, Berlin. ISBN 978-1-84800-906-6.

EXTENDED ABSTRACTS

- Victor Cheung, Diane Watson, **Jo Vermeulen**, Mark Hancock, and Stacey D. Scott. Overcoming Interaction Barriers in Large Public Displays Using Personal Devices. In *Adjunct Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '14 Adjunct, 6 pages, 2014. ACM.
- Yannick Bernaerts, Matthias Druwé, Sebastiaan Steensels, **Jo Vermeulen**, and Johannes Schöning. The Office Smartwatch – Development and Design of a Smartwatch App to Digitally Augment Interactions in an Office Environment. In *Adjunct Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14 Adjunct, pages 41–44, 2014. ACM. ISBN 978-1-4503-2903-3.
- Linsey Raymaekers, **Jo Vermeulen**, Kris Luyten, and Karin Coninx. Game of Tones: Learning to Play Songs on a Piano Using Projected Instructions and Games. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '14, pages 411–414, 2014. ACM. ISBN 978-1-4503-2474-8.
- **Jo Vermeulen**. Improving Intelligibility and Control in Ubicomp. In *Adjunct Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, Ubicomp '10 Adjunct, pages 485–488, 2010. ACM. ISBN 978-1-4503-0283-8.
- **Jo Vermeulen**, Geert Vanderhulst, Kris Luyten, and Karin Coninx. Answering Why and Why Not Questions in Ubiquitous Computing. In *Proceedings of the 11th ACM International Conference on Ubiquitous Computing – Adjunct Papers*, Ubicomp '09 Adjunct, pages 210–213, 2009.
- Kris Luyten, Jan Meskens, **Jo Vermeulen**, and Karin Coninx. Meta-GUI-Builders: Generating Domain-Specific Interface Builders for Multi-Device User Interface Creation. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '08, pages 3189–3194, 2014. ACM. ISBN 978-1-60558-012-8.

WORKSHOP PAPERS

- **Jo Vermeulen**, Kris Luyten, Karin Coninx and Nicolai Marquardt. Addressing Challenges in Crowded Proxemics-Aware Installations. In *Proceedings of the Workshop on SocialNUI: Social Perspectives in Natural User Interfaces*, DIS '14 workshop. 4 pages.

- **Jo Vermeulen**, Kris Luyten, and Karin Coninx. The Feedforward Torch. In *Proceedings of the Second Workshop on Intelligibility and Control in Pervasive Computing*, Pervasive '12 workshop. 2 pages.
- Johanna Renny Octavia, Kris Luyten, **Jo Vermeulen**, Benji Mommen, and Karin Coninx. Exploring Psycho-physiological Measures for the Design and Behavior of Intelligent, Socially-Aware Avatars in Ubicomp Environments. In *Proceedings of Brain, Body and Bytes: Psychophysiological User Interaction*, CHI '10 workshop. 4 pages.
- **Jo Vermeulen**, Ruben Thys, Kris Luyten and Karin Coninx. Making Bits and Atoms Talk Today: A Practical Architecture for Smart Object Interaction. In *Proceedings DIPSO 2007: 1st International Workshop on Design and Integration Principles for Smart Objects*, Ubicomp '07 workshop. pp. 331–336.
- **Jo Vermeulen**, Kris Luyten, Karin Coninx, and Ruben Thys. Tangible Mashups: Exploiting Links between the Physical and Virtual World. In *Proceedings of WoSSIoT'07: 1st International Workshop on System Support for the Internet of Things*, EuroSys '07 workshop. 4 pages.
- Kris Luyten, **Jo Vermeulen**, and Karin Coninx. Constraint Adaptability of Multi-Device User Interfaces. In *Proceedings of The Many Faces of Consistency in Cross-Platform Design*, CHI '06 workshop. pp. 40–45.
- Kris Luyten, Karin Coninx, **Jo Vermeulen**, Mieke Haesen, and Luk Vloemans. ImogI: Take control over a context-aware electronic mobile guide for museums. In *Workshop on HCI in Mobile Guides*, MobileHCI '04 workshop, 2004.

B

USER STUDIES

This appendix lists the different documents that were used during user studies.

B.1 SITUATED GLYPHS

This section lists the questionnaires used for the user study of *Situated Glyphs*, as described in Section 4.3.4.

Participant No:
Situation: Embedded – System (EmS)

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree
1. It was simple to use this system.	**1**	**2**	**3**	**4**	**5**
2. I was able to complete the navigation tasks quickly using this system.	**1**	**2**	**3**	**4**	**5**
3. I felt comfortable using this system.	**1**	**2**	**3**	**4**	**5**
4. It was easy to learn to use this system.	**1**	**2**	**3**	**4**	**5**
5. The interface of this system is pleasant.	**1**	**2**	**3**	**4**	**5**
6. I liked using the interface of this system.	**1**	**2**	**3**	**4**	**5**
7. Overall, I am satisfied with this system.	**1**	**2**	**3**	**4**	**5**

	Very low	Low	Undecided	High	Very High
A. Mental demand. How mentally demanding was the task?	**1**	**2**	**3**	**4**	**5**
B. Physical Demand. How physically demanding was the task?	**1**	**2**	**3**	**4**	**5**
B. Effort. How hard did you have to work to accomplish your level of performance?	**1**	**2**	**3**	**4**	**5**
C. Frustration level. How insecure, discouraged, irritated, stressed, and annoyed were you?	**1**	**2**	**3**	**4**	**5**

Positive Aspects: _____

Negative Aspects: _____

Comments: _____

Interview

1. Age

2. Sex

3. How would you define your experience with computers?
    a. None
    b. Poor
    c. Medium
    d. High
    e. Expert

4. How would you define your experience with mobile phones?
    a. None
    b. Poor
    c. Medium
    d. High
    e. Expert

5. Do you own a mobile phone (iPhone/Others) ?

6. Please rank your preference between having the information projected in the environment, and between having the information available on the mobile phone (iPhone).

Interaction technique	**Preference (1st, 2nd)**
iPhone	
Projection	

7. Please rank your preference between having information being provided automatically, and having to explicitly ask for information.

Interaction technique	**Preference (1st, 2nd)**
System-initiated	
User-initiated	

8. Discussion on the four different interaction techniques.

9. When an activity is finished, do you think it is useful to show a confirmation *after* the activity has been completed?
   - Yes / No
   - Comments: _____
     _____
     _____
     _____

10. To you think it is useful to get information about the next activity to be performed *before* you start it?
    - Yes / No
    - Comments: _____
      _____
      _____
      _____

11. Did you notice information about the current activity was sometimes shown *during* the time you were performing the activity? Do you think this is useful?
    - Noticed? Yes / No
    - Useful? Yes / No
    - Comments: _____
      _____
      _____
      _____

12. What would you do differently regarding *when* to show information about an activity?

13. When would you like to present different types of information?

Information type	**When**
Identity	Before – During – After
Relationship	Before – During – After
Instructions	Before – During – After
Explanations	Before – During – After
Confirmations	Before – During – After

- Comments: _____
  _____
  _____
  _____
  _____
  _____
  _____

14. Something you liked from the experiment?

15. Something that you dislike from the experiment?

16. Any further comments or suggestions?

B.2 THE FEEDFORWARD TORCH

This section lists the questionnaires used for the user study of the *Feedforward Torch*, as described in Section 5.6.4.

Gebruikersstudie:
Vragenlijst

Hartelijk dank voor uw deelname aan de gebruikersstudie van de Feedforward Torch. Omdat uw bevindingen van dit systeem van uiterst belang zijn voorhet onderzoek zouden we u willen vragen onderstaande vragenlijst in tevullen. De vragenlijst bestaat uit drie delen, waar u bij het eerste deelbeperkte persoonlijke informatie dient op te geven. Het tweede deel bestaat uit gestructureerde vragen op een 5-punt schaal. Er dient een score gegeven te worden per vraag die varieert tussen 1 en 5, waarbij score 1 betekent dat u het volledig eens bent met de stelling, en score 5 dat u het helemaal niet eens bent met de opgegeven stelling. Tot slot is er nog een deel 3 dat bestaat uit een aantal open vragen eventueel aangevuld met een 5-punt schaal. Bij zowel de open vragen als de eventuele opmerkingen bij de gestructureerde vragen zouden wij u willen vragen om uw eerlijke mening neer te schrijven. Op basis van zowel uw positieve als uw negatieve bevindingen kunnen we afleiden wat de sterke en zwakke eigenschappen zijn van het systeem en kan er bepaald worden of dit systeem een potentiële toekomst heeft.

A. Persoonlijke vragen

Leeftijd:

Geslacht: Man/Vrouw

Opleiding:

Beroep:

B. Gestructureerde vragen

1. Het was eenvoudig om dit systeem te gebruiken.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

2. Ik was, dankzij het systeem, in staat om de taken en scenario's snel te voltooien.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

3. Ik voelde me comfortabel met dit systeem.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

4. Het was gemakkelijk om dit systeem aan te leren.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

5. De informatie (zoals tekst, iconen, afbeeldingen, ...) aangeboden door dit systeem is duidelijk.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

6. De informatie, aangeboden om mij te begeleiden met het uitvoeren van de taken en scenario's, is effectief.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

7. De weergave van de informatie op het scherm van het systeem is duidelijk.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

8. Het systeem is een hulpmiddel in onbekende situaties.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

9. Ik geef de voorkeur aan grafische visualisaties boven tekstuele verklaringen.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

10. Over het algemeen ben ik tevreden over dit systeem.

Volledig mee eens 1 2 3 4 5 Helemaal niet mee eens

Opmerkingen:

C. Open vragen

1. Vindt u de gebruikte animaties (bv: projectiescherm dat naar beneden komt) een meerwaarde bij acties die zich afspelen over de tijd?
   Motiveer uw antwoord.

Uw mening gesitueerd in een 5-punt schaal:

Zeer positief 1 2 3 4 5 Zeer negatief

2. Wanneer verkiest u om gebruik te maken van de projectie en wanneer niet?

3. Zou u het een meerwaarde vinden dat de getoonde informatie rechtstreeks weergegeven werd op het camerabeeld van de smartphone (zie figuur D.1)?
Motiveer uw antwoord.

Uw mening gesitueerd in een 5-punt schaal:
Zeer positief 1 2 3 4 5 Zeer negatief

4. Zou u het systeem in de praktijk gebruiken?
Motiveer uw antwoord.

5. Positieve aspecten van het systeem?

6. Negatieve aspecten van het systeem?

7. Andere opmerkingen?

Camerabeeld van de smartphone uitgebreid met extra informatie

B.3 THE VISIBLE COMPUTER

This section lists the questionnaires used for the user study of *The Visible Computer*, as described in Section 6.3.5.

Gebruikerstest: The Visible Computer

Schrijf extra uitleg bij je antwoord wanneer mogelijk.

Q1: Ik begrijp hoe ik de visualisatie van de omgeving kan gebruiken.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q2: Ik vind de visualisatie van de omgeving gemakkelijk te gebruiken.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q3: De visualisatie van de omgeving was eenvoudig te begrijpen.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q4: De visualisatie was wat ik wilde weten.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q5: Ik begrijp hoe ik de cancel functionaliteit moet gebruiken.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q6: Ik vond de cancel functionaliteit gemakkelijk te gebruiken.

Helemaal akkoord	1	2	3	4	5	Helemaal niet akkoord

Q7: Ik vind dat deze technieken nuttig zijn om gebruikers te laten begrijpen wat er gebeurt in een “slimme” omgeving, en hun erover controle te laten uitoefenen.

Helemaal akkoord	1	2	3	4	5	Helemaal niet akkoord

Q8: Ik was verward door de visualisaties.

Helemaal akkoord	1	2	3	4	5	Helemaal niet akkoord

B.4 PERVASIVECRYSTAL

This section lists the questionnaires used for the user study of *PervasiveCrystal*, as described in Section 7.6.

Gebruikerstest Why en Why Not vragen

Q1: Ik begrijp hoe ik de Why en Why Not vragen moet gebruiken

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q2: Ik vond de Why en Why Not vragen gemakkelijk te gebruiken.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q3: De antwoorden van de Why en Why Not vragen waren eenvoudig te begrijpen.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q4: De antwoorden van de Why en Why Not vragen waren wat ik wilde weten.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q5: Ik begrijp hoe ik de controle functionaliteit (undo, do en “how can I …”) moet gebruiken.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q6: Ik vond de controle functionaliteit (undo, do en “how can I …”) gemakkelijk te gebruiken.

Helemaal akkoord 1 2 3 4 5 Helemaal niet akkoord

Q7: Ik vind dat deze technieken nuttig zijn om gebruikers te laten begrijpen wat er gebeurt in een intelligente omgeving, en hun erover controle te laten uitoefenen.

Helemaal akkoord	1	2	3	4	5	Helemaal niet akkoord

Q8: Ik was niet verward door de vragen of het tabel.

Helemaal akkoord	1	2	3	4	5	Helemaal niet akkoord

C

NEDERLANDSTALIGE SAMENVATTING

In dit doctoraatsproefschrift, bestudeer ik het ontwerp van gebruikersinterfaces voor ubiquitous computing (ubicomp) omgevingen die ondersteuning bieden voor *intelligibility* en *control*. In de literatuur werden reeds een aantal veel voorkomende moeilijkheden geïdentificeerd omtrent interactie met ubicomp systemen, en in het bijzonder contextgevoelige systemen, waaronder het gebrek aan *begrip* over waarom het systeem op een bepaalde manier reageert en het gebrek aan mogelijkheden tot *interventie* wanneer het systeem fouten maakt. Deze problemen kunnen het vertrouwen van gebruikers in het systeem beïnvloeden alsook hun gevoel van controle over dat systeem. Uiteindelijk kan dit zelfs leiden tot gebruikers die volledig afzien van interactie met het computersysteem. Het is noodzakelijk om deze problemen op te lossen om ubicomp technologie op een naadloze manier in ons dagelijks leven te kunnen integreren.

In de literatuur werden twee algemene principes voorgesteld die contextgevoelige systemen dienen te ondersteunen om deze problemen aan te pakken: intelligibility en control. Intelligibility is de mogelijkheid van een systeem om zichzelf en zijn gedrag op een begrijpbare manier aan zijn gebruikers voor te stellen. Control daarentegen gaat over het bieden van de mogelijkheid om gebruikers te laten tussenkomen en het systeem te corrigeren. Hoewel een aantal technieken om intelligibility en control te voorzien reeds onderzocht werden, is het nog niet duidelijk op welke manier ubicomp onderzoekers, ontwikkelaars en interaction designers gebruikersinterfaces kunnen ontwerpen die ondersteuning bieden voor intelligibility en control.

Dit doctoraatsproefschrift is een *design space exploration* van mogelijke technieken om intelligibility en control te ondersteunen. Het doel van deze design space exploration is tweevoudig: (1) het voorstellen van algemene principes voor intelligibility en control die gebruikt kunnen worden in een breed spectrum van ubicomp applicaties, en (2) het gidsen van designers om verschillende manieren te verkennen om intelligibility en control te ondersteunen. In het bijzonder stelt dit proefschrift de volgende drie originele bijdragen voor.

Ten eerste, stel ik een design space voor die verschillende ontwerpbeslissingen voorstelt waarmee designers geconfronteerd worden bij het ontwerp van interfaces om intelligibility en control te ondersteunen. Deze design space omvat zes dimensies en kan gebruikt worden als een analytisch instrument om verschillende technieken te classificeren en te vergelijken. Tevens ondersteunt de design space designers in het verkennen van alternatieve designs.

Ten tweede, introduceer ik algemene principes en interactietechnieken die toegepast kunnen worden in een breed scala aan ubicomp applicaties. In dit doctoraatsproefschrift, verken ik de *timing* dimensie uit bovenstaande design space in detail.

Ik stel drie algemene technieken voor die gebruikt kunnen worden op verschillende momenten tijdens de interactie: *feedforward* (vóór acties), *slow-motion feedback* (tijdens acties) en *why questions* (na acties).

Als derde en laatste bijdrage, bespreek ik een gedetailleerde case study omtrent het ondersteunen van intelligibility en control voor *proxemic interactions*. Proxemic interactions stelt een specifieke soort van contextgevoelige applicaties voor, waarbij apparaten rekening houden met factoren zoals de afstand, locatie en identiteit van personen en apparaten in de buurt. Deze case study kan dienen als inspiratie voor designers en onderzoekers die intelligibility en control willen voorzien voor hun eigen ubicomp applicaties. In het bijzonder, stellen we in deze case study het gebruik van een secundaire ondersteunende floor display voor die gebruikers helpt tijdens het interageren met het primaire display. Dit floor display informeert gebruikers over de tracking status, geeft interactiemogelijkheden aan, en nodigt gebruikers uit voor en gidst hen doorheen interactie met het primaire display.

BIBLIOGRAPHY

Gregory D. Abowd. Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Syst. J.*, 38(4):508–530, December 1999. ISSN 0018-8670. (Cited on page 12.)

Gregory D. Abowd. What next, ubicomp?: Celebrating an intellectual disappearing act. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 31–40, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1224-0. (Cited on page 8.)

Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, 7(1):29–58, March 2000. ISSN 1073-0516. (Cited on pages 9 and 17.)

Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wirel. Netw.*, 3(5):421–433, October 1997. ISSN 1022-0038. (Cited on page 10.)

Corporate Act-Net Consortium. The active database management system manifesto: a rulebase of adbms features. *SIGMOD Rec.*, 25(3):40–49, 1996. ISSN 0163-5808. (Cited on page 134.)

Eytan Adar, Desney S. Tan, and Jaime Teevan. Benevolent deception in human computer interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1863–1872, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1899-0. (Cited on page 93.)

Petr Aksenov, Kris Luyten, and Karin Coninx. O brother, where art thou located?: raising awareness of variability in location tracking for users of location-based pervasive applications. *Journal of Location Based Services*, 6(4):211–233, 2012. (Cited on page 32.)

Bashar Altakouri, Gerd Kortuem, Agnes Grunerbl, Kai Kunze, and Paul Lukowicz. The benefit of activity recognition for mobile phone based nursing documentation: A wizard-of-oz study. In *Wearable Computers (ISWC), 2010 International Symposium on*, pages 1–4, Oct 2010. (Cited on pages 51 and 56.)

Michelle Annett, Tovi Grossman, Daniel Wigdor, and George Fitzmaurice. Medusa: A proximity-aware multi-touch tabletop. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 337–346, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. (Cited on page 152.)

Stavros Antifakos, Nicky Kern, Bernt Schiele, and Adrian Schwaninger. Towards improving trust in context-aware systems by displaying system confidence. In *Proceedings of the 7th International Conference on Human Computer Interaction with*

Mobile Devices & Services, MobileHCI '05, pages 9–14, New York, NY, USA, 2005. ACM. ISBN 1-59593-089-2. (Cited on page 33.)

Mark Assad, David J. Carmichael, Judy Kay, and Bob Kummerfeld. PersonisAD: Distributed, active, scrutable model framework for context-aware services. In *Proceedings of the 5th International Conference on Pervasive Computing (Pervasive 2007)*, volume 4480 of *Lecture Notes in Comput. Sci.*, pages 55–72. Springer, 2007. ISBN 978-3-540-72036-2. (Cited on pages 17, 20, and 132.)

Takenori Atsumi. Feedforward control using sampled-data polynomial for track seeking in hard disk drives. *Industrial Electronics, IEEE Transactions on*, 56(5):1338–1346, May 2009. ISSN 0278-0046. (Cited on page 74.)

Thomas Augsten, Konstantin Kaefer, René Meusel, Caroline Fetzer, Dorian Kanitz, Thomas Stoff, Torsten Becker, Christian Holz, and Patrick Baudisch. Multitoe: High-precision interaction with back-projected floors based on high-resolution multi-touch input. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 209–218, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0271-5. (Cited on pages 147 and 153.)

Ronald Azuma. A survey of augmented reality. *Presence*, 6(4):355–385, 1997. (Cited on pages 99 and 115.)

Till Ballendat, Nicolai Marquardt, and Saul Greenberg. Proxemic interaction: Designing for a proximity and orientation-aware environment. In *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, pages 121–130, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0399-6. (Cited on pages xxii, 146, 149, 150, 152, 153, 155, and 160.)

Jakob E. Bardram. A novel approach for creating activity-aware applications in a hospital environment. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II*, INTERACT '09, pages 731–744, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03657-6. (Cited on pages 52 and 70.)

Jakob E. Bardram, Thomas R. Hansen, Martin Mogensen, and Mads Soegaard. Experiences from real-world deployment of context-aware technologies in a hospital environment. In *Proceedings of the 8th International Conference on Ubiquitous Computing*, UbiComp'06, pages 369–386, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-39634-9, 978-3-540-39634-5. (Cited on page 51.)

Louise Barkhuus and Anind K. Dey. Is context-aware computing taking control away from the user? three levels of interactivity examined. In *Proceedings of the 5th International Conference on Ubiquitous Computing*, volume 2864 of *Lecture Notes in Comput. Sci.*, pages 149–156. Springer, 2003. ISBN 3-540-20301-X. (Cited on pages 18 and 19.)

Olivier Bau and Wendy E. Mackay. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium*

on User interface software and technology, pages 37–46, Monterey, CA, USA, 2008. ACM. ISBN 978-1-59593-975-3. (Cited on pages xix, 34, 38, 41, 44, 77, 78, 91, 94, 95, and 111.)

Patrick Baudisch, Desney Tan, Maxime Collomb, Dan Robbins, Ken Hinckley, Maneesh Agrawala, Shengdong Zhao, and Gonzalo Ramos. Phosphor: Explaining transitions in the user interface using afterglow effects. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, pages 169–178, New York, NY, USA, 2006. ACM. ISBN 1-59593-313-1. (Cited on page 125.)

BBC News. Millions tricked by 'scareware'. `http://news.bbc.co.uk/2/hi/technology/8313678.stm`, October 2009. [Online; accessed 07-August-2014]. (Cited on page 93.)

Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 15–22, New York, NY, USA, 2004. ACM. ISBN 1-58113-867-9. (Cited on page 2.)

Genevieve Bell and Paul Dourish. Yesterday's tomorrows: Notes on ubiquitous computing's dominant vision. *Personal Ubiquitous Comput.*, 11(2):133–143, January 2007. ISSN 1617-4909. (Cited on page 7.)

Victoria Bellotti and Keith Edwards. Intelligibility and accountability: human considerations in context-aware systems. *Hum.-Comput. Interact.*, 16(2):193–212, 2001. ISSN 0737-0024. (Cited on pages 1, 2, 14, 16, 19, 20, 25, 26, 27, 31, 34, 74, 75, 77, 78, 127, 156, and 175.)

Victoria Bellotti and Abigail Sellen. Design for privacy in ubiquitous computing environments. In *Proceedings of the Third Conference on European Conference on Computer-Supported Cooperative Work*, ECSCW'93, pages 77–92, Norwell, MA, USA, 1993. Kluwer Academic Publishers. ISBN 0-7923-2447-1. (Cited on page 175.)

Victoria Bellotti, Maribeth Back, W. Keith Edwards, Rebecca E. Grinter, Austin Henderson, and Cristina Lopes. Making sense of sensing systems: five questions for designers and researchers. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*, CHI '02, pages 415–422, Minneapolis, Minnesota, USA, 2002. ACM. ISBN 1-58113-453-3. (Cited on pages 1, 12, 13, 18, 105, 106, 115, 152, and 180.)

Jacques Bertin. *Semiology of graphics: diagrams, networks, maps*. University of Wisconsin Press, 1983. (Cited on page 52.)

Gilbert Beyer, Vincent Binder, Nina Jäger, and Andreas Butz. The puppeteer display: Attracting and actively shaping the audience with an interactive public banner display. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 935–944, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2902-6. (Cited on pages 152, 163, and 171.)

Hugh Beyer. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. ISBN 1558604111. (Cited on page 56.)

Jon Bird, Daniel Harrison, and Paul Marshall. The challenge of maintaining interest in a large-scale public floor display. In *Proceedings of the Experiencing Interactivity in Public Spaces Workshop*, EIPS '13, 2013. (Cited on page 164.)

Jan Borchers, Meredith Ringel, Joshua Tyler, and Armando Fox. Stanford interactive workspaces: a framework for physical and graphical user interface prototyping. *Wireless Communications, IEEE*, 9(6):64–69, Dec 2002. ISSN 1536-1284. (Cited on page 11.)

S. Boring, S. Greenberg, J. Vermeulen, J. Dostal, and N. Marquardt. The dark patterns of proxemic sensing. *Computer*, 47(8):56–60, Aug 2014. ISSN 0018-9162. (Cited on page 147.)

Alan Bränzel, Christian Holz, Daniel Hoffmann, Dominik Schmidt, Marius Knaust, Patrick Lühne, René Meusel, Stephan Richter, and Patrick Baudisch. Gravityspace: Tracking users and their poses in a smart room using a pressure-sensing floor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 725–734, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1899-0. (Cited on pages 153, 154, and 168.)

Harry Brignull. Dark patterns: Deception vs. honesty in ui design. `http://alistapart.com/article/dark-patterns-deception-vs.-honesty-in-ui-design`, November 2011. [Online; accessed 07-August-2014]. (Cited on page 93.)

Harry Brignull and Yvonne Rogers. Enticing people to interact with large public displays in public spaces. In *Human-Computer Interaction INTERACT '03: IFIP TC13 International Conference on Human-Computer Interaction, 1st-5th September 2003, Zurich, Switzerland*, INTERACT '03. IOS Press, 2003. ISBN 1-58603-363-8. (Cited on pages 146, 150, 151, 158, and 170.)

John Seely Brown and Susan E. Newman. Issues in cognitive and social ergonomics: from our house to bauhaus. *Hum.-Comput. Interact.*, 1(4):359–391, 1985. (Cited on page 16.)

Frederik Brudy, David Ledo, Saul Greenberg, and Andreas Butz. Is anyone looking? mitigating shoulder surfing on public displays through awareness and protection. In *Proceedings of The International Symposium on Pervasive Displays*, PerDis '14, pages 1:1–1:6, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2952-1. (Cited on page 176.)

Andrea Bunt, Matthew Lount, and Catherine Lauzon. Are explanations always important?: A study of deployed, low-cost intelligent interactive systems. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces*, IUI

'12, pages 169–178, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1048-2. (Cited on page 21.)

Stefano Burigat and Luca Chittaro. Pedestrian navigation with degraded gps signal: Investigating the effects of visualizing position uncertainty. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 221–230, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0541-9. (Cited on page 32.)

Andreas Butz and Antonio Krüger. Applying the peephole metaphor in a mixed-reality room. *IEEE Comput. Graph. Appl.*, 26(1):56–63, 2006. ISSN 0272-1716. (Cited on page 122.)

Bill Buxton. Integrating the periphery and context: A new taxonomy of telematics. In *Proceedings of Graphics Interface*, GI '95, pages 239–246, 1995a. (Cited on pages xvii and 11.)

Bill Buxton. Proximal sensing: Supporting context sensitive. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, UIST '95, pages 169–170, New York, NY, USA, 1995b. ACM. ISBN 0-89791-709-X. (Cited on pages 10 and 17.)

Xiang Cao, Clifton Forlines, and Ravin Balakrishnan. Multi-user interaction using handheld projectors. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 43–52, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-679-0. (Cited on pages 99 and 115.)

Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. A morphological analysis of the design space of input devices. *ACM Trans. Inf. Syst.*, 9(2):99–122, April 1991. ISSN 1046-8188. (Cited on page 37.)

Bay-Wei Chang and David Ungar. Animation: From cartoons to the user interface. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, UIST '93, pages 45–55, New York, NY, USA, 1993. ACM. ISBN 0-89791-628-X. (Cited on page 112.)

Han Chen, Rahul Sukthankar, Grant Wallace, and Kai Li. Scalable alignment of large-format multi-projector displays using camera homography trees. In *Proc. VIS '02*, pages 339–346. IEEE Computer Society, 2002. ISBN 0-7803-7498-3. (Cited on pages 115 and 119.)

Herman Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68(342):361–368, 1973. (Cited on page 52.)

Victor Cheung and Stacey D. Scott. Investigating attraction and engagement of animation on large interactive walls in public settings. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '13, pages 381–384, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2271-3. (Cited on pages 151 and 162.)

Victor Cheung, Diane Watson, Jo Vermeulen, Mark Hancock, and Stacey D. Scott. Overcoming interaction barriers in large public displays using personal devices. In *Proceedings of the 2014 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '14, New York, NY, USA, 2014. ACM. (Cited on pages 151 and 175.)

Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a context-aware electronic tourist guide: Some issues and experiences. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, pages 17–24, New York, NY, USA, 2000. ACM. ISBN 1-58113-216-6. (Cited on page 10.)

Keith Cheverst, Nigel Davies, Keith Mitchell, and Christos Efstratiou. Using context as a crystal ball: Rewards and pitfalls. *Personal Ubiquitous Comput.*, 5(1):8–11, 2001. ISSN 1617-4909. (Cited on pages 1, 15, and 16.)

Keith Cheverst, Keith Mitchell, and Nigel Davies. Exploring context-aware information push. *Personal Ubiquitous Comput.*, 6(4):276–281, January 2002. ISSN 1617-4909. (Cited on page 19.)

Keith Cheverst, Hee Eon Byun, Dan Fitton, Corina Sas, Chris Kray, and Nicolas Villar. Exploring issues of user model transparency and proactive behaviour in an office environment control system. *User Modeling and User-Adapted Interaction*, 15(3-4):235–273, 2005. ISSN 0924-1868. (Cited on pages 17, 20, 26, 32, 33, 35, 39, 47, 49, 132, and 143.)

Mei C. Chuah and Stephen G. Eick. Information rich glyphs for software management data. *IEEE Comput. Graph. Appl.*, 18(4):24–29, July 1998. ISSN 0272-1716. (Cited on page 52.)

Herbert H. Clark and Susan E. Brennan. Grounding in communication. *Perspectives on socially shared cognition*, 13(1991):127–149, 1991. (Cited on page 25.)

Marcelo Coelho and Jamie Zigelbaum. Shape-changing interfaces. *Personal and Ubiquitous Computing*, 15(2):161–173, 2011. ISSN 1617-4909. (Cited on page 82.)

Diane J. Cook, Michael Youngblood, Edwin O. Heierman, III, Karthik Gopalratnam, Sira Rao, Andrey Litvin, and Farhan Khawaja. Mavhome: An agent-based smart home. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, PERCOM '03, pages 521–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1893-1. (Cited on pages 12 and 19.)

Jeremy R. Cooperstock, Sidney S. Fels, William Buxton, and Kenneth C. Smith. Reactive environments. *Commun. ACM*, 40(9):65–73, September 1997. ISSN 0001-0782. (Cited on pages 9, 10, 12, 13, 16, 17, and 19.)

Joëlle Coutaz. Meta-user interfaces for ambient spaces. In *Proceedings of the 5th International Conference on Task Models and Diagrams for Users Interface Design*, TAMODIA'06, pages 1–15, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-70815-5. (Cited on pages 18, 20, 37, and 41.)

Joëlle Coutaz, Laurence Nigay, Daniel Salber, Ann Blandford, Jon May, and Richard M. Young. Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. In *Human-Computer Interaction, INTERACT '95, IFIP TC13 Interantional Conference on Human-Computer Interaction*, INTERACT '95, pages 115–120. Chapman & Hall, 1995. ISBN 0-412-71790-5. (Cited on page 176.)

Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Commun. ACM*, 48(3):49–53, March 2005. ISSN 0001-0782. (Cited on page 17.)

Joëlle Coutaz, Sybille Caffiau, Alexandre Demeure, and James L. Crowley. Early lessons from the development of spok, an end-user development environment for smart homes. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, pages 895–902, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3047-3. (Cited on page 178.)

Henriette Cramer, Vanessa Evers, Satyan Ramlal, Maarten Someren, Lloyd Rutledge, Natalia Stash, Lora Aroyo, and Bob Wielinga. The effects of transparency on trust in and acceptance of a content-based art recommender. *User Modeling and User-Adapted Interaction*, 18(5):455–496, November 2008. ISSN 0924-1868. (Cited on pages 32 and 132.)

Allen Cypher. Eager: Programming repetitive tasks by example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 33–39, New York, NY, USA, 1991. ACM. ISBN 0-89791-383-3. (Cited on page 75.)

Allen Cypher, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky, editors. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, USA, 1993. ISBN 0-262-03213-9. (Cited on page 36.)

Nicholas Sheep Dalton. TapTiles: LED-based Floor Interaction. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '13, pages 165–174, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2271-3. (Cited on page 154.)

Scott Davidoff. Ubicomp 2012. *IEEE Pervasive Computing*, 11(3):84–88, July 2012. ISSN 1536-1268. (Cited on page 8.)

David Dearman, Alex Varshavsky, Eyal De Lara, and Khai N. Truong. An exploration of location error estimation. In *Proceedings of the 9th International Conference on Ubiquitous Computing*, UbiComp '07, pages 181–198, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74852-6. (Cited on page 32.)

Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1): 4–7, 2001. ISSN 1617-4909. (Cited on page 9.)

Anind K. Dey and Jennifer Mankoff. Designing mediation for context-aware applications. *ACM Trans. Comput.-Hum. Interact.*, 12(1):53–80, 2005. ISSN 1073-0516. (Cited on page 35.)

Anind K. Dey and Alan Newberger. Support for context-aware intelligibility and control. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 859–868, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7. (Cited on pages 13, 25, 34, 35, and 47.)

Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16(2):97–166, December 2001. ISSN 0737-0024. (Cited on pages 14, 34, 133, and 146.)

Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. A CAPpella: Programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 33–40, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. (Cited on pages 36, 42, 47, and 49.)

Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. iCAP: Interactive Prototyping of Context-Aware Applications. In *Pervasive Computing*, volume 3968 of *Lecture Notes in Computer Science*, pages 254–271. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-33894-9. (Cited on pages 35, 42, 47, 115, 118, and 178.)

Tom Djajadiningrat, Kees Overbeeke, and Stephan Wensveen. But How, Donald, Tell Us How?: On the Creation of Meaning in Interaction Design Through Feedforward and Inherent Feedback. In *Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '02, pages 285–291, New York, NY, USA, 2002. ACM. ISBN 1-58113-515-7. (Cited on pages 18, 75, 76, 77, 78, 79, 81, 83, 84, 85, 87, 94, and 95.)

Tom Djajadiningrat, Stephan Wensveen, Joep Frens, and Kees Overbeeke. Tangible products: redressing the balance between appearance and action. *Personal Ubiquitous Comput.*, 8(5):294–309, 2004. (Cited on page 87.)

Paul Dourish. Developing a reflective model of collaborative systems. *ACM Trans. Comput.-Hum. Interact.*, 2(1):40–63, March 1995. ISSN 1073-0516. (Cited on pages 17, 20, and 21.)

Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, 2004. ISSN 1617-4909. (Cited on pages 15, 17, and 25.)

Paul Dourish and Sara Bly. Portholes: Supporting awareness in a distributed work group. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 541–547, New York, NY, USA, 1992. ACM. ISBN 0-89791-513-5. (Cited on pages xvii, 10, and 11.)

Pierre Dragicevic, Anastasia Bezerianos, Waqas Javed, Niklas Elmqvist, and Jean-Daniel Fekete. Temporal distortion for animated transitions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2009–2018, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0228-9. (Cited on page 112.)

W. Keith Edwards and Rebecca E. Grinter. At home with ubiquitous computing: Seven challenges. In *Proceedings of the 3rd International Conference on Ubiquitous Computing*, UbiComp '01, pages 256–272, London, UK, 2001. Springer-Verlag. ISBN 3-540-42614-0. (Cited on page 18.)

Thomas Erickson. Some problems with the notion of context-aware computing. *Commun. ACM*, 45(2):102–104, 2002. ISSN 0001-0782. (Cited on pages 1, 14, and 19.)

Jesus Favela, Monica Tentori, Luis A. Castro, Victor M. Gonzalez, Elisa B. Moran, and Ana I. Martínez-García. Activity recognition for context-aware hospital applications: Issues and opportunities for the deployment of pervasive networks. *Mob. Netw. Appl.*, 12(2-3):155–171, March 2007. ISSN 1383-469X. (Cited on pages 51 and 70.)

Steven Feiner, Blair Macintyre, and Dorée Seligmann. Knowledge-based augmented reality. *Commun. ACM*, 36(7):53–62, July 1993. ISSN 0001-0782. (Cited on pages 99 and 115.)

George W. Fitzmaurice, Hiroshi Ishii, and William A. S. Buxton. Bricks: Laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 442–449, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1. (Cited on page 37.)

Dustin Freeman, Hrvoje Benko, Meredith Ringel Morris, and Daniel Wigdor. Shadowguides: visualizations for in-situ learning of multi-touch and whole-hand gestures. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pages 165–172, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-733-2. (Cited on pages 38, 41, 77, 94, 95, and 152.)

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0-201-63361-2. (Cited on page 169.)

Alfonso García Frey, Gaëlle Calvary, and Sophie Dupuy-Chessa. Users need your models!: Exploiting design models for explanations. In *Proceedings of the 26th Annual BCS Interaction Specialist Group Conference on People and Computers*, BCS-HCI '12, pages 79–88, Swinton, UK, UK, 2012. British Computer Society. (Cited on page 132.)

Alfonso García Frey, Gaëlle Calvary, Sophie Dupuy-Chessa, and Nadine Mandran. Model-Based Self-explanatory UIs for Free, but Are They Valuable? In Paula Kotzé, Gary Marsden, Gitte Lindgaard, Janet Wesson, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2013*, volume 8119 of *Lecture Notes in Computer Science*, pages 144–161. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40476-4. (Cited on page 132.)

William W. Gaver. Technology affordances. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, CHI '91, pages 79–84, New Orleans, Louisiana, United States, 1991. ACM. ISBN 0-89791-383-3. (Cited on pages 81, 82, 83, 85, 92, 94, and 95.)

Hans Gellersen, Carl Fischer, Dominique Guinard, Roswitha Gostner, Gerd Kortuem, Christian Kray, Enrico Rukzio, and Sara Streng. Supporting device discovery and spontaneous interaction with spatial references. *Personal Ubiquitous Comput.*, 13(4):255–264, May 2009. ISSN 1617-4909. (Cited on pages 146 and 152.)

James J. Gibson. The theory of affordances. In Robert E. Shaw and John Bransford, editors, *Perceiving, acting and knowing: toward an ecological psychology*, pages 67–82. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1977. ISBN 0-470-99014-7. (Cited on pages 75, 79, 80, 83, 85, 89, and 92.)

James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton, Mifflin and Company, Boston, MA, USA, 1979. ISBN 0-89859-959-8. (Cited on pages 75, 79, 82, 83, 85, 89, and 92.)

Kazjon Grace, Rainer Wasinger, Christopher Ackad, Anthony Collins, Oliver Dawson, Richard Gluga, Judy Kay, and Martin Tomitsch. Conveying interactivity at an interactive public information display. In *Proceedings of the 2nd ACM International Symposium on Pervasive Displays*, PerDis '13, pages 19–24, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2096-2. (Cited on page 152.)

Saul Greenberg. Context as a dynamic construct. *Hum.-Comput. Interact.*, 16(2): 257–268, December 2001. ISSN 0737-0024. (Cited on pages 1, 14, 15, 16, 17, 18, and 19.)

Saul Greenberg, Nicolai Marquardt, Till Ballendat, Rob Diaz-Marino, and Miaosen Wang. Proxemic interactions: The new ubicomp? *interactions*, 18(1):42–50, January 2011. ISSN 1072-5520. (Cited on pages xxii, 145, 146, 147, 155, and 160.)

Saul Greenberg, Sebastian Boring, Jo Vermeulen, and Jakub Dostal. Dark patterns in proxemic interactions: A critical perspective. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 523–532, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2902-6. (Cited on pages 50, 146, and 150.)

Shirley Gregor and Izak Benbasat. Explanations from intelligent systems: Theoretical foundations and implications for practice. *MIS Q.*, 23(4):497–530, December 1999. ISSN 0276-7783. (Cited on pages 32, 127, and 131.)

Kaj Grønbæk, Ole S. Iversen, Karen Johanne Kortbek, Kaspar Rosengreen Nielsen, and Louise Aagaard. Igamefloor: A platform for co-located collaborative games. In *Proceedings of the International Conference on Advances in Computer Entertainment Technology*, ACE '07, pages 64–71, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-640-0. (Cited on pages 153 and 154.)

Tovi Grossman and George Fitzmaurice. Toolclips: An investigation of contextual video assistance for functionality understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1515–1524, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. (Cited on page 179.)

Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A survey of software learnability: Metrics, methodologies and guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 649–658, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7. (Cited on page 179.)

Edward T. Hall. A system for the notation of proxemic behavior. *American Anthropologist*, 65(5):1003–1026, 1963. ISSN 1548-1433. (Cited on page 146.)

Edward T. Hall. *The hidden dimension*, volume 1990. Anchor Books New York, 1969. (Cited on page 145.)

Chris Harrison, Zhiquan Yeo, and Scott E. Hudson. Faster progress bars: Manipulating perceived duration with visual augmentations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1545–1548, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. (Cited on page 93.)

Chris Harrison, Hrvoje Benko, and Andrew D. Wilson. Omnitouch: Wearable multitouch interaction everywhere. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 441–450, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. (Cited on page 99.)

Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (task load index): Results of empirical and theoretical research. In Peter A. Hancock and Najmedin Meshkati, editors, *Human Mental Workload*, volume 52 of *Advances in Psychology*, pages 139–183. North-Holland, 1988. (Cited on page 63.)

Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 145–154, New York, NY, USA, 2007a. ACM. ISBN 978-1-59593-593-9. (Cited on page 178.)

Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R. Klemmer. Programming by a sample: Rapidly creating web applications with d.mix. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 241–250, New York, NY, USA, 2007b. ACM. ISBN 978-1-59593-679-0. (Cited on page 178.)

Rex Hartson. Cognitive, physical, sensory, and functional affordances in interaction design. *Behaviour & Information Technology*, 22(5):315, 2003. ISSN 0144-929X. (Cited on pages xix, 81, 83, 84, 85, 86, 87, 88, 89, 90, and 91.)

Rex Hartson. personal email communication, 2010. (Cited on pages 84 and 90.)

Luke Hespanhol, Martin Tomitsch, Oliver Bown, and Miriama Young. Using Embodied Audio-visual Interaction to Promote Social Encounters Around Large Media Façades. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 945–954, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2902-6. (Cited on pages 152 and 153.)

Valentin Heun, Anette von Kapri, and Pattie Maes. Perifoveal display: Combining foveal and peripheral vision in one visualization. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 1150–1155, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1224-0. (Cited on page 171.)

Clint Heyer. Investigations of ubicomp in the oil and gas industry. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, Ubicomp '10, pages 61–64, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-843-8. (Cited on page 52.)

Ken Hinckley. Synchronous gestures for multiple persons and computers. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, UIST '03, pages 149–158, New York, NY, USA, 2003. ACM. ISBN 1-58113-636-6. (Cited on page 146.)

Ken Hinckley, Jeff Pierce, Mike Sinclair, and Eric Horvitz. Sensing techniques for mobile interaction. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, UIST '00, pages 91–100, New York, NY, USA, 2000. ACM. ISBN 1-58113-212-3. (Cited on pages 11 and 16.)

Ken Hinckley, Gonzalo Ramos, Francois Guimbretiere, Patrick Baudisch, and Marc Smith. Stitching: Pen gestures that span multiple displays. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 23–31, New York, NY, USA, 2004. ACM. ISBN 1-58113-867-9. (Cited on page 146.)

Paul Holleis, Enrico Rukzio, Friderike Otto, and Albrecht Schmidt. Privacy and curiosity in mobile interactions with public displays. In *Proceedings of the CHI 2007 Workshop on Mobile Spatial Interaction*, 2007. (Cited on page 151.)

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. (Cited on page 74.)

Steven Houben and Christian Weichel. Overcoming interaction blindness through curiosity objects. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 1539–1544, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1952-2. (Cited on pages 147 and 151.)

Elaine M. Huang, Anna Koster, and Jan Borchers. Overcoming assumptions and uncovering practices: When does the public really look at public displays? In *Proceedings of the 6th International Conference on Pervasive Computing*, Pervasive '08, pages 228–243, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-79575-9. (Cited on page 147.)

Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. Direct manipulation interfaces. *Hum.-Comput. Interact.*, 1(4):311–338, December 1985. ISSN 0737-0024. (Cited on page 12.)

M. Iwasaki and N. Matusi. Robust speed control of IM with torque feedforward control. *Industrial Electronics, IEEE Transactions on*, 40(6):553–560, Dec 1993. ISSN 0278-0046. (Cited on page 74.)

Robert J. K. Jacob. What you look at is what you get: Eye movement-based interaction techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 11–18, New York, NY, USA, 1990. ACM. ISBN 0-201-50932-6. (Cited on page 13.)

Wendy Ju. *The design of implicit interactions*. PhD thesis, Stanford University, June 2008. (Cited on pages 10, 13, 24, and 28.)

Wendy Ju and Larry Leifer. The design of implicit interactions: Making interactive systems less obnoxious. *Design Issues*, 24(3):72–84, 2008. (Cited on page 28.)

Wendy Ju, Brian A. Lee, and Scott R. Klemmer. Range: Exploring implicit interaction through electronic whiteboard design. In *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, CSCW '08, pages 17–26, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-007-4. (Cited on pages 28, 34, 35, 38, 40, 42, 46, 47, 105, 122, 124, 146, 152, 153, and 170.)

Marko Jurmu, Masaki Ogawa, Sebastian Boring, Jukka Riekki, and Hideyuki Tokuda. Waving to a touch interface: Descriptive field study of a multipurpose multimodal public display. In *Proceedings of the 2nd ACM International Symposium on Pervasive Displays*, PerDis '13, pages 7–12, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2096-2. (Cited on pages 33, 146, 147, 150, 152, and 159.)

Victor Kaptelinin and Bonnie Nardi. Affordances in HCI: toward a mediated action perspective. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 967–976, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. (Cited on pages 81, 82, and 83.)

Evan Karatzas. Proximity lab: Studies in physical-computational interface and self-directed user experience. Master's thesis, Massachusetts College of Art, 2005. `http://www.proximity-lab.com/work.php?id=11`. (Cited on pages 152 and 153.)

Fahim Kawsar, Tatsuo Nakajima, and Kaori Fujinami. Deploy spontaneously: Supporting end-users in building and enhancing a smart home. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, pages 282–291, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-136-1. (Cited on pages 36 and 178.)

Fahim Kawsar, Jo Vermeulen, Kevin Smith, Kris Luyten, and Gerd Kortuem. Exploring the design space for situated glyphs to support dynamic work environments.

In *Pervasive Computing*, volume 6696 of *Lecture Notes in Computer Science*, pages 70–78. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21725-8. (Cited on pages xi, 51, 52, and 56.)

Judy Kay, Bob Kummerfeld, and Piers Lauder. Managing private user models and shared personas. In *UM03 Workshop on User Modeling for Ubiquitous Computing*, pages 1–11, 2003. (Cited on pages 17 and 20.)

John F. Kelley. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Inf. Syst.*, 2(1):26–41, January 1984. ISSN 1046-8188. (Cited on pages 58, 96, and 120.)

Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, Thad Starner, and Wendy Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture*, CoBuild '99, pages 191–198, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66596-X. (Cited on page 12.)

Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, places, things: web presence for the real world. *Mob. Netw. Appl.*, 7(5):365–376, 2002. ISSN 1383-469X. (Cited on page 11.)

Mikkel Baun Kjærgaard and Kay Weckemann. Posq: Unsupervised fingerprinting and visualization of gps positioning quality. In Martin Gris and Guang Yang, editors, *Mobile Computing, Applications, and Services*, volume 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 176–194. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-29335-1. (Cited on page 32.)

Andrew J. Ko and Brad A. Myers. Designing the whyline: A debugging interface for asking questions about program behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 151–158, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. (Cited on page 127.)

Andrew J. Ko and Brad A. Myers. Finding causes of program output with the java whyline. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1569–1578, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7. (Cited on pages 127 and 128.)

Gerd Kortuem, Fahim Kawsar, Phil Scholl, Michael Beigl, Adalberto L. Simeone, and Kevin Smith. A miniaturized display network for situated glyphs. In *Adjunct Proceedings of the 9th International Conference on Pervasive Computing*, 2011. Demo. (Cited on page 54.)

Christian Kray, Michael Rohs, Jonathan Hook, and Sven Kratz. Group coordination and negotiation through spatial proximity regions around mobile devices

on augmented tabletops. In *Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on*, IEEE Tabletop '08, pages 1–8, Oct 2008. (Cited on page 146.)

Todd Kulesza, Weng-Keen Wong, Simone Stumpf, Stephen Perona, Rachel White, Margaret M. Burnett, Ian Oberst, and Andrew J. Ko. Fixing the program my computer learned: Barriers for end users, challenges for the machine. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, pages 187–196, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-168-2. (Cited on pages 27, 32, 36, 39, 133, and 179.)

Todd Kulesza, Simone Stumpf, Weng-Keen Wong, Margaret M. Burnett, Stephen Perona, Andrew Ko, and Ian Oberst. Why-oriented end-user debugging of naive bayes text classification. *ACM Trans. Interact. Intell. Syst.*, 1(1):2:1–2:31, October 2011. ISSN 2160-6455. (Cited on page 179.)

Todd Kulesza, Simone Stumpf, Margaret Burnett, Sherry Yang, Irwin Kwan, and Weng-Keen Wong. Too much, too little, or just right? ways explanations impact end users' mental models. In *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*, pages 3–10, Sept 2013. (Cited on pages 20, 36, and 133.)

Gordon Kurtenbach and William Buxton. User learning and performance with marking menus. In *Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence*, CHI '94, pages 258–264, New York, NY, USA, 1994. ACM. ISBN 0-89791-650-6. (Cited on page 77.)

Gordon P. Kurtenbach, Abigail J. Sellen, and William A. S. Buxton. An empirical evaluation of some articulatory and cognitive aspects of marking menus. *Hum.-Comput. Interact.*, 8(1):1–23, 1993. (Cited on pages xix, 44, 76, 77, and 92.)

Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. Community enhanced tutorials: Improving tutorials with multiple demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1779–1788, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1899-0. (Cited on page 179.)

Samuel Lapkin, Tracy Levett-Jones, Helen Bellchambers, and Ritin Fernandez. Effectiveness of patient simulation manikins in teaching clinical reasoning skills to undergraduate nursing students: A systematic review. *Clinical Simulation in Nursing*, 6(6):e207–e222, 2010. ISSN 1876-1399. (Cited on page 60.)

Diana Laurillard. *Rethinking university teaching: a framework for the effective use of educational technologies*. Routledge, 1993. (Cited on page 80.)

Hendrik Lemelson, Thomas King, and Wolfgang Effelsberg. A study on user acceptance of error visualization techniques. In *Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Mobiquitous '08, pages 53:1–53:6, ICST, Brussels, Belgium, Belgium, 2008.

ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-27-1. (Cited on pages 32 and 33.)

Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. Coscripter: Automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1719–1728, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. (Cited on page 178.)

James R. Lewis. IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *Int. J. Hum.-Comput. Interact.*, 7(1):57–78, January 1995. ISSN 1044-7318. (Cited on page 63.)

Brian Y. Lim and Anind K. Dey. Assessing demand for intelligibility in context-aware applications. In *Proceedings of the 11th International Conference on Ubiquitous Computing*, Ubicomp '09, pages 195–204, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-431-7. (Cited on pages 32, 38, 128, 133, 143, and 174.)

Brian Y. Lim and Anind K. Dey. Toolkit to support intelligibility in context-aware applications. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, Ubicomp '10, pages 13–22, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-843-8. (Cited on pages 2, 26, 27, 32, 38, 39, 78, 133, 143, and 144.)

Brian Y. Lim and Anind K. Dey. Design of an intelligible mobile context-aware application. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 157–166, New York, NY, USA, 2011a. ACM. ISBN 978-1-4503-0541-9. (Cited on pages 32, 33, and 174.)

Brian Y. Lim and Anind K. Dey. Investigating intelligibility for uncertain context-aware applications. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 415–424, New York, NY, USA, 2011b. ACM. ISBN 978-1-4503-0630-0. (Cited on pages 20 and 174.)

Brian Y. Lim, Anind K. Dey, and Daniel Avrahami. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2119–2128, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7. (Cited on pages 26, 32, 128, 132, and 174.)

Kris Luyten, Karin Coninx, Jo Vermeulen, Mieke Haesen, and Luk Vloemans. ImogI: Take control over a context-aware electronic mobile guide for museums. In *Workshop on HCI in Mobile Guides, in conjunction with the 6th International Conference on Human Computer Interaction with Mobile Devices and Services*, 2004. (Cited on page 10.)

Thomas M. Mann. Visualization of www-search results. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, DEXA '99, pages 264–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0281-4. (Cited on page 52.)

Nicolai Marquardt, Tom Gross, Sheelagh Carpendale, and Saul Greenberg. Revealing the invisible: Visualizing the location and event flow of distributed physical devices. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '10, pages 41–48, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-841-4. (Cited on pages 115 and 116.)

Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. The proximity toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 315–326, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. (Cited on pages 146 and 169.)

Nicolai Marquardt, Till Ballendat, Sebastian Boring, Saul Greenberg, and Ken Hinckley. Gradual engagement: Facilitating information exchange between digital devices as a function of proximity. In *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '12, pages 31–40, New York, NY, USA, 2012a. ACM. ISBN 978-1-4503-1209-7. (Cited on pages 123, 124, 146, and 152.)

Nicolai Marquardt, Ken Hinckley, and Saul Greenberg. Cross-device interaction via micro-mobility and f-formations. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 13–22, New York, NY, USA, 2012b. ACM. ISBN 978-1-4503-1580-7. (Cited on page 146.)

Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. Communitycommands: Command recommendations for software applications. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, pages 193–202, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-745-5. (Cited on page 179.)

Joseph F. McCarthy, David W. McDonald, Suzanne Soroczak, David H. Nguyen, and Al M. Rashid. Augmenting the social space of an academic conference. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, pages 39–48, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. (Cited on page 11.)

Joanna McGrenere and Wayne Ho. Affordances: Clarifying and evolving a concept. In *Proceedings of the Graphics Interface 2000 Conference*, pages 179—186, 2000. (Cited on pages 81, 82, 83, 84, and 85.)

Peter H. Meckl and Warren P. Seering. Feedforward control techniques to achieve fast settling time in robots. In *American Control Conference, 1986*, pages 1913 –1918, june 1986. (Cited on page 74.)

Sarah Mennicken, Jo Vermeulen, and Elaine M. Huang. From today's augmented houses to tomorrow's smart homes: New directions for home automation research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 105–115, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2968-2. (Cited on pages 12, 19, 49, and 178.)

Daniel Michelis and Jörg Müller. The audience funnel: Observations of gesture based interaction with multiple large displays in a city center. *International Journal of Human-Computer Interaction*, 27(6):562–579, 2011. (Cited on page 151.)

David Molyneaux, Hans Gellersen, Gerd Kortuem, and Bernt Schiele. Cooperative augmentation of smart objects with projector-camera systems. In *Proceedings of the 9th International Conference on Ubiquitous Computing*, UbiComp '07, pages 501–518, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74852-6. (Cited on pages 99 and 115.)

David Molyneaux, Shahram Izadi, David Kim, Otmar Hilliges, Steve Hodges, Xiang Cao, Alex Butler, and Hans Gellersen. Interactive environment-aware handheld projectors for pervasive computing spaces. In *Proceedings of the 10th International Conference on Pervasive Computing*, Pervasive'12, pages 197–215, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-31204-5. (Cited on pages 99 and 115.)

Michael Mozer. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, pages 110–114, 1998. (Cited on page 12.)

Florian Mueller, Sophie Stellmach, Saul Greenberg, Andreas Dippon, Susanne Boll, Jayden Garner, Rohit Khot, Amani Naseem, and David Altimira. Proxemics play: Understanding proxemics for designing digital play experiences. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 533–542, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2902-6. (Cited on page 157.)

Jörg Müller, Juliane Exeler, Markus Buzeck, and Antonio Krüger. Reflectivesigns: Digital signs that adapt to audience attention. In *Proceedings of the 7th International Conference on Pervasive Computing*, Pervasive '09, pages 17–24, Berlin, Heidelberg, 2009a. Springer-Verlag. ISBN 978-3-642-01515-1. (Cited on page 146.)

Jörg Müller, Dennis Wilmsmann, Juliane Exeler, Markus Buzeck, Albrecht Schmidt, Tim Jay, and Antonio Krüger. Display blindness: The effect of expectations on attention towards digital signage. In *Proceedings of the 7th International Conference on Pervasive Computing*, Pervasive '09, pages 1–8, Berlin, Heidelberg, 2009b. Springer-Verlag. ISBN 978-3-642-01515-1. (Cited on pages 147, 151, and 175.)

Jörg Müller, Florian Alt, Daniel Michelis, and Albrecht Schmidt. Requirements and design space for interactive public displays. In *Proceedings of the International Conference on Multimedia*, MM '10, pages 1285–1294, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-933-6. (Cited on pages 147, 150, 151, and 159.)

Jörg Müller, Robert Walter, Gilles Bailly, Michael Nischt, and Florian Alt. Looking glass: A field study on noticing interactivity of a shop window. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 297–306, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. (Cited on pages 146, 152, 171, and 175.)

Jörg Müller, Gilles Bailly, Thor Bossuyt, and Niklas Hillgren. Mirrortouch: Combining touch and mid-air gestures for public displays. In *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices & Services*, MobileHCI '14, pages 319–328, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3004-6. (Cited on page 171.)

Brad A. Myers, David A. Weitzman, Andrew J. Ko, and Duen H. Chau. Answering why and why not questions in user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 397–406, New York, NY, USA, 2006. ACM. ISBN 1-59593-372-7. (Cited on pages 32, 39, 127, and 128.)

Bonnie A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-computer Interaction*. Massachusetts Institute of Technology, Cambridge, MA, USA, 1995. ISBN 0-262-14058-6. (Cited on page 15.)

Mark W. Newman, Shahram Izadi, W. Keith Edwards, Jana Z. Sedivy, and Trevor F. Smith. User interfaces when and where they are needed: An infrastructure for recombinant computing. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology*, UIST '02, pages 171–180, New York, NY, USA, 2002. ACM. ISBN 1-58113-488-6. (Cited on page 12.)

Jakob Nielsen. Noncommand user interfaces. *Commun. ACM*, 36(4):83–99, April 1993. ISSN 0001-0782. (Cited on pages 10, 11, 12, 41, 42, and 75.)

Donald A. Norman. *The Psychology Of Everyday Things*. Basic Books, New York, USA, June 1988. ISBN 0465067093. (Cited on pages xix, 21, 74, 75, 76, 79, 81, 83, 84, 85, 87, 88, 89, 90, and 92.)

Donald A. Norman. The 'problem' with automation: Inappropriate feedback and interaction, not 'over-automation'. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 327(1241):585–593, 1990. (Cited on pages 19, 21, and 23.)

Donald A. Norman. Affordance, conventions, and design. *interactions*, 6(3):38–43, 1999. (Cited on pages 75 and 82.)

Donald A. Norman. *The Design of Everyday Things*. Basic Books, September 2002. ISBN 0465067107. (Cited on page 21.)

Donald A. Norman. The way I see it: Signifiers, not affordances. *interactions*, 15(6): 18–19, November 2008. ISSN 1072-5520. (Cited on pages xix, 74, and 76.)

Donald A. Norman. *The Design of Future Things*. Basic Books, New York, first trade paper edition edition edition, May 2009. ISBN 9780465002283. (Cited on pages 1, 24, and 25.)

Donald A. Norman. Natural user interfaces are not natural. *interactions*, 17(3):6–10, May 2010. ISSN 1072-5520. (Cited on pages 13 and 118.)

Donald A. Norman. *Living With Complexity*. MIT Press, 2011. ISBN 9780262014861. (Cited on page 73.)

Donald A. Norman. personal email communication, July 2013a. (Cited on page 89.)

Donald A. Norman. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books, New York, revised edition, November 2013b. ISBN 9780465050659. (Cited on pages xvii, xix, 4, 13, 17, 18, 21, 22, 23, 25, 26, 27, 73, 74, 75, 78, 88, 89, 92, and 127.)

Kenton O'Hara. Interactivity and non-interactivity on tabletops. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2611–2614, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. (Cited on page 150.)

Timo Ojala, Vassilis Kostakos, Hannu Kukka, Tommi Heikkinen, Tomas Linden, Marko Jurmu, Simo Hosio, Fabio Kruger, and Daniele Zanni. Multipurpose interactive public displays in the wild: Three years later. *Computer*, 45(5):42–49, May 2012. ISSN 0018-9162. (Cited on pages 146, 147, 149, 151, and 175.)

OSGi Alliance. *OSGi service platform, release 3*. IOS Press, Inc., 2003. (Cited on page 134.)

Ovgu Ozturk, Tomoaki Matsunami, Yasuhiro Suzuki, Toshihiko Yamasaki, and Kiyoharu Aizawa. Real-time tracking of humans and visualization of their future footsteps in public indoor environments. *Multimedia Tools Appl.*, 59(1):65–88, July 2012. ISSN 1380-7501. (Cited on page 153.)

Susan Palmiter and Jay Elkerton. Animated demonstrations for learning procedural computer-based tasks. *Hum.-Comput. Interact.*, 8(3):193–216, September 1993. ISSN 0737-0024. (Cited on page 45.)

John F. Pane and Brad A. Myers. Tabular and textual methods for selecting objects from a group. In *Proceedings of the 2000 IEEE International Symposium on Visual Languages (VL'00)*, VL '00, pages 157–, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0840-5. (Cited on page 118.)

Richard Chulwoo Park, Hyunjae Lee, Hwan Kim, and Woohun Lee. The previewable switch: A light switch with feedforward. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 191–194, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2902-6. (Cited on page 92.)

Thomas Pederson. *From Conceptual Links to Causal Relations – Physical-Virtual Artefacts in Mixed-Reality Space*. PhD thesis, Umeå University, Faculty of Science and Technology, Computing Science, 2003. (Cited on page 54.)

Claudio S. Pinhanez. The everywhere displays projector: A device to create ubiquitous graphical interfaces. In *Proceedings of the 3rd International Conference on Ubiquitous Computing*, UbiComp '01, pages 315–331, London, UK, 2001. Springer-Verlag. ISBN 3-540-42614-0. (Cited on pages 98, 115, and 119.)

Zachary Pousman and John Stasko. A taxonomy of ambient information systems: Four patterns of design. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '06, pages 67–74, New York, NY, USA, 2006. ACM. ISBN 1-59593-353-0. (Cited on page 53.)

Thorsten Prante, Carsten Röcker, Norbert Streitz, Richard Stenzel, Carsten Magerkurth, Daniel Van Alphen, and Daniela Plewe. Hello. Wall–beyond ambient displays. In *Adjunct Proceedings of the Fifth International Conference on Ubiquitous Computing*, Ubicomp '03 Adjunct, pages 277–278, 2003. (Cited on page 146.)

Sriranjan Rasakatla and K. Madhava Krishna. Way-go torch: An intelligent flash light. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 1765–1771, Dec 2013. (Cited on page 99.)

Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 179–188, New York, NY, USA, 1998. ACM. ISBN 0-89791-999-8. (Cited on page 115.)

Ramesh Raskar, Jeroen van Baar, Paul Beardsley, Thomas Willwacher, Srinivas Rao, and Clifton Forlines. ilamps: Geometrically aware and self-configuring projectors. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 809–818, New York, NY, USA, 2003. ACM. ISBN 1-58113-709-5. (Cited on pages 98, 99, 102, 115, and 119.)

Ramesh Raskar, Paul Beardsley, Jeroen van Baar, Yao Wang, Paul Dietz, Johnny Lee, Darren Leigh, and Thomas Willwacher. Rfig lamps: Interacting with a self-describing world via photosensing wireless tags and projectors. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 406–415, New York, NY, USA, 2004. ACM. (Cited on pages 98 and 115.)

Kasim Rehman, Frank Stajano, and George Coulouris. Interfacing with the invisible computer. In *Proceedings of the Second Nordic Conference on Human-computer Interaction*, NordiCHI '02, pages 213–216, New York, NY, USA, 2002. ACM. ISBN 1-58113-616-1. (Cited on pages 17 and 74.)

Kasim Rehman, Frank Stajano, and George Coulouris. Visually interactive location-aware computing. In *Proceedings of the 7th International Conference on Ubiquitous Computing*, volume 3660 of *Lecture Notes in Computer Science*, pages 177–194. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-28760-5. (Cited on pages 33, 38, 42, 115, 121, and 153.)

Tom Rodden, Andy Crabtree, Terry Hemmings, Boriana Koleva, Jan Humble, Karl-Petter Åkesson, and Pär Hansson. Configuring the ubiquitous home. In *Cooperative Systems Design, Scenario-Based Design of Collaborative Systems*, COOP '04, pages 227–242. IOS, 2004. ISBN 1-58603-422-7. (Cited on pages 35, 47, 49, and 178.)

Matt Rogers. 4.0 software update for the Nest Thermostat and app. https://nest.com/blog/2013/11/15/4-software-update-for-nest-thermostat-and-app/, November 2013. [Online; accessed 25-July-2014]. (Cited on page 34.)

Yvonne Rogers. New theoretical approaches for human-computer interaction. *Annual Review of Information Science and Technology*, 38(1):87–143, 2004. ISSN 1550-8382. (Cited on page 2.)

Yvonne Rogers. Moving on from weiser's vision of calm computing: Engaging ubicomp experiences. In *Proceedings of the 8th International Conference on Ubiquitous Computing*, UbiComp'06, pages 404–421, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-39634-9, 978-3-540-39634-5. (Cited on pages 1 and 14.)

Yvonne Rogers. Interaction design gone wild: Striving for wild theory. *interactions*, 18(4):58–62, July 2011. ISSN 1072-5520. (Cited on pages 168 and 174.)

Yvonne Rogers, William R. Hazlewood, Paul Marshall, Nick Dalton, and Susanna Hertrich. Ambient influence: Can twinkly lights lure and abstract representations trigger behavioral change? In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, Ubicomp '10, pages 261–270, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-843-8. (Cited on pages 153 and 162.)

Stephanie Rosenthal, Shaun K. Kane, Jacob O. Wobbrock, and Daniel Avrahami. Augmenting on-screen instructions with micro-projected guides: When it works, and when it fails. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, Ubicomp '10, pages 203–212, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-843-8. (Cited on page 99.)

Enrico Rukzio and Paul Holleis. Projector phone interactions: Design space and survey. In *Proceedings of the Workshop on Coupled Display Visual Interfaces (PPD '10), in conjunction with AVI 2010*, 2010. (Cited on page 102.)

Enrico Rukzio, John Hamard, Chie Noda, and Alexander De Luca. Visualization of uncertainty in context aware mobile applications. In *Proceedings of the 8th Conference on Human-computer Interaction with Mobile Devices and Services*, MobileHCI '06, pages 247–250, New York, NY, USA, 2006. ACM. ISBN 1-59593-390-5. (Cited on page 33.)

Enrico Rukzio, Paul Holleis, and Hans Gellersen. Personal projectors for pervasive computing. *IEEE Pervasive Computing*, 11(2):30–37, April 2012. ISSN 1536-1268. (Cited on pages 68, 96, 99, and 102.)

Dan Saffer. *Designing for Interaction: Creating Innovative Applications and Devices (2nd Edition)*. New Riders Press, 2 edition, August 2009. ISBN 0321643399. (Cited on page 78.)

Dan Saffer. *Microinteractions: Full Color Edition: Designing with Details*. O'Reilly Media, Sebastopol, CA, November 2013. ISBN 9781491945926. (Cited on page 16.)

Christian Sandor, Alex Olwal, Blaine Bell, and Steven Feiner. Immersive mixed-reality configuration of hybrid user interfaces. In *Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '05, pages 110–113, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2459-1. (Cited on page 115.)

Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE Computer Society, Dec 1994. ISBN 978-0-7695-3451-0. (Cited on pages 8, 9, 10, and 146.)

Albrecht Schmidt. Implicit human computer interaction through context. *Personal Technologies*, 4(2-3):191–199, 2000. ISSN 0949-2054. (Cited on page 10.)

Dominik Schmidt, Raf Ramakers, Esben W. Pedersen, Johannes Jasper, Sven Köhler, Aileen Pohl, Hannes Rantzsch, Andreas Rau, Patrick Schmidt, Christoph Sterz, Yanina Yurchenko, and Patrick Baudisch. Kickables: Tangibles for feet. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 3143–3152, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. (Cited on pages 147 and 153.)

Mindy Seto, Stacey Scott, and Mark Hancock. Investigating menu discoverability on a digital tabletop in a public setting. In *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '12, pages 71–80, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1209-7. (Cited on page 151.)

Mary Shaw. The role of design spaces. *Software, IEEE*, 29(1):46–50, Jan 2012. ISSN 0740-7459. (Cited on page 37.)

Thomas B Sheridan and Raja Parasuraman. Human-automation interaction. *Reviews of human factors and ergonomics*, 1(1):89–129, 2005. (Cited on page 19.)

Ben Shneiderman. A second path to hci innovation: Generative theories tied to user needs. In *Proceedings of the CHI 2006 Workshop: "What is the Next Generation of Human-Computer Interaction"*, 2006. (Cited on page 2.)

Rajinder Sodhi, Hrvoje Benko, and Andrew Wilson. Lightguide: Projected visualizations for hand movement guidance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 179–188, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. (Cited on page 153.)

Norbert Streitz, Achilles Kameas, and Irene Mavrommati. *The Disappearing Computer: Interaction Design, System Infrastructures and Applications for Smart Environments*. Springer-Verlag, 2007. ISBN 3540727256. (Cited on page 113.)

Lucy A. Suchman. *Plans and situated actions: the problem of human-machine communication*. Cambridge University Press, 1987. ISBN 0-521-33137-4. (Cited on page 15.)

Rahul Sukthankar, Robert G. Stockton, and Matthew D. Mullin. Smarter presentations: exploiting homography in camera-projector systems. In *Proceedings of the Eight IEEE International Conference on Computer Vision*, volume 1 of *ICCV '01*, pages 247–253, 2001. (Cited on page 119.)

Desney S. Tan, Ivan Poupyrev, Mark Billinghurst, Hirokazu Kato, Holger Regenbrecht, and Nobuji Tetsutani. On-demand, in-place help for augmented reality environments. In *Proceedings of the 3rd International Conference on Ubiquitous Computing*, Ubicomp '01, 2001. (Cited on page 115.)

Alex S. Taylor. Intelligence in context. In *Proceedings of the International Symposium on Intelligent Environments*, April 2006. (Cited on pages 14, 15, and 17.)

Joe Tullio, Anind K. Dey, Jason Chalecki, and James Fogarty. How it works: A field study of non-technical users interacting with an intelligent system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 31–40, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. (Cited on pages 13, 20, and 132.)

Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. Animation: Can it facilitate? *Int. J. Hum.-Comput. Stud.*, 57(4):247–262, October 2002. ISSN 1071-5819. (Cited on page 45.)

Davy Vanacken, Alexandre Demeure, Kris Luyten, and Karin Coninx. Ghosts in the interface: Meta-user interface visualizations as guides for multi-touch interaction. In *Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on*, pages 81–84, Oct 2008. (Cited on pages 41, 77, 94, 95, 152, and 179.)

Geert Vanderhulst. *Development and Deployment of Interactive Pervasive Applications for Ambient Intelligent Environments*. PhD thesis, Hasselt University, 2010. (Cited on pages xxi, 134, and 135.)

Geert Vanderhulst, Kris Luyten, and Karin Coninx. Middleware for ubiquitous service-oriented spaces on the web. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, AINAW '07, pages 1001–1006, 2007. (Cited on page 120.)

Geert Vanderhulst, Kris Luyten, and Karin Coninx. ReWiRe: Creating interactive pervasive systems that cope with changing environments by rewiring. In *Intelligent Environments, 2008 IET 4th International Conference on*, pages 1–8, July 2008. (Cited on pages 128 and 133.)

Jo Vermeulen. Improving intelligibility and control in ubicomp. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing – Adjunct Papers*, Ubicomp '10 Adjunct, pages 485–488, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0283-8. (Cited on page 36.)

Jo Vermeulen, Jonathan Slenders, Kris Luyten, and Karin Coninx. I Bet You Look Good on the Wall: Making the Invisible Computer Visible. In *Ambient Intelligence*, volume 5859 of *Lecture Notes in Computer Science*, pages 196–205. Springer Berlin Heidelberg, 2009a. ISBN 978-3-642-05407-5. (Cited on pages xi, 33, 42, 43, 45, 99, 105, and 113.)

Jo Vermeulen, Geert Vanderhulst, Kris Luyten, and Karin Coninx. Answering why and why not questions in ubiquitous computing. In *Proceedings of the 11th ACM International Conference on Ubiquitous Computing – Adjunct Papers*, Ubicomp '09 Adjunct, pages 210–213, 2009b. (Cited on pages 27, 32, and 141.)

Jo Vermeulen, Geert Vanderhulst, Kris Luyten, and Karin Coninx. PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications. In *Intelligent Environments (IE), 2010 Sixth International Conference on*, pages 271–276, July 2010. (Cited on pages xi, 27, 32, 35, 39, 40, 42, 46, and 126.)

Jo Vermeulen, Fahim Kawsar, Adalberto L. Simeone, Gerd Kortuem, Kris Luyten, and Karin Coninx. Informing the design of situated glyphs for a care facility. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, VLHCC '12, pages 89–96, Sept 2012a. (Cited on pages xi, 51, and 71.)

Jo Vermeulen, Kris Luyten, and Karin Coninx. Understanding complex environments with the feedforward torch. In *Ambient Intelligence*, volume 7683 of *Lecture Notes in Computer Science*, pages 312–319. Springer Berlin Heidelberg, 2012b. ISBN 978-3-642-34897-6. (Cited on pages xi, 42, and 46.)

Jo Vermeulen, Kris Luyten, and Karin Coninx. Intelligibility required: How to make us look smart again. In *Proceedings of the 10th Romanian Conference on Human-Computer Interaction*, ROCHI '13, 2013a. (Cited on page 36.)

Jo Vermeulen, Kris Luyten, Elise van den Hoven, and Karin Coninx. Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1931–1940, New York, NY, USA, 2013b. ACM. ISBN 978-1-4503-1899-0. (Cited on pages xi, 18, 27, 34, 78, and 88.)

Jo Vermeulen, Kris Luyten, Karin Coninx, and Nicolai Marquardt. The design of slow-motion feedback. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 267–270, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2902-6. (Cited on pages xi, 24, 34, and 105.)

Roel Vertegaal. Attentive user interfaces. *Commun. ACM*, 46(3):30–33, March 2003. ISSN 0001-0782. (Cited on page 124.)

Sami Vihavainen, Antti Oulasvirta, and Risto Sarvas. "I can't lie anymore!": The implications of location automation for mobile social applications. In *Mobile and Ubiquitous Systems: Networking Services, MobiQuitous, 2009. MobiQuitous '09. 6th Annual International*, pages 1–10, July 2009. (Cited on pages 16, 18, and 19.)

Yon Visell, Alvin Law, and Jeremy R. Cooperstock. Touch is everywhere: Floor surfaces as ambient haptic interfaces. *EEE Trans. Haptics*, 2(3):148–159, July 2009. ISSN 1939-1412. (Cited on pages 154 and 170.)

Daniel Vogel and Ravin Balakrishnan. Interactive public ambient displays: Transitioning from implicit to explicit, public to personal, interaction with multiple users. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST '04, pages 137–146, New York, NY, USA, 2004. ACM. ISBN 1-58113-957-8. (Cited on pages 146, 150, 152, 153, 159, 160, and 175.)

Robert Walter, Gilles Bailly, and Jörg Müller. Strikeapose: Revealing mid-air gestures on public displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 841–850, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1899-0. (Cited on pages 152 and 171.)

Miaosen Wang, Sebastian Boring, and Saul Greenberg. Proxemic peddler: A public advertising display that captures and preserves the attention of a passerby. In *Proceedings of the 2012 International Symposium on Pervasive Displays*, PerDis '12, pages 3:1–3:6, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1414-5. (Cited on page 151.)

Roy Want. Introduction to ubiquitous computing. In John Krumm, editor, *Ubiquitous Computing Fundamentals*, chapter 1. Chapman & Hall/CRC, 1st edition, 2009. ISBN 1420093606, 9781420093605. (Cited on page 8.)

Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, January 1992. ISSN 1046-8188. (Cited on page 10.)

Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging physical and virtual worlds with electronic tags. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 370–377, New York, NY, USA, 1999. ACM. ISBN 0-201-48559-1. (Cited on page 11.)

Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3 edition, 2012. ISBN 9780123814647, 9780123814654. (Cited on page 52.)

Alan J. Wecker, Joel Lanir, Tsvi Kuflik, and Oliviero Stock. Pathlight: Supporting navigation of small groups in the museum context. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 569–574, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0541-9. (Cited on page 99.)

Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991. (Cited on pages xvii, 1, 7, 8, 9, 113, 114, and 145.)

Mark Weiser. Creating the invisible interface: (invited talk). In *Proc. UIST '94*, page 1. ACM, 1994. ISBN 0-89791-657-3. (Cited on page 113.)

Mark Weiser and John Seely Brown. Designing calm technology. http://www.ubiq.com/hypertext/weiser/calmtech/calmtech.htm, December 1995. [Online; accessed 14-July-2014]. (Cited on pages 1 and 8.)

Mark Weiser and John Seely Brown. The coming age of calm technolgy. In Peter J. Denning and Robert M. Metcalfe, editors, *Beyond Calculation*, pages 75–85. Copernicus, New York, NY, USA, 1997. ISBN 0-38794932-1. (Cited on page 8.)

Mark Weiser, Rich Gold, and John Seely Brown. The origins of ubiquitous computing research at parc in the late 1980s. *IBM Syst. J.*, 38(4):693–696, December 1999. ISSN 0018-8670. (Cited on pages 7 and 8.)

Evan Welbourne, Magdalena Balazinska, Gaetano Borriello, and James Fogarty. Specification and verification of complex location events with panoramic. In *Proceedings of the 8th International Conference on Pervasive Computing*, Pervasive '10, pages 57–75, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-12653-7, 978-3-642-12653-6. (Cited on pages 115 and 132.)

S. A. G. Wensveen, J. P. Djajadiningrat, and C. J. Overbeeke. Interaction frogger: a design framework to couple action and function through feedback and feedforward. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '04, pages 177–184, Cambridge, MA, USA, 2004. ACM. ISBN 1-58113-787-7. (Cited on pages 79, 84, 85, and 91.)

Stephan Wensveen. *A Tangibility Approach to Affective Interaction*. PhD thesis, TU Delft, 2005. (Cited on pages xix, 18, 73, 79, 80, 81, 82, 83, 84, 85, 92, 94, and 95.)

Stephan Wensveen. personal email communication, 2010. (Cited on page 81.)

Sean White, Levi Lister, and Steven Feiner. Visual hints for tangible gestures in augmented reality. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, pages 1–4, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 978-1-4244-1749-0. (Cited on page 115.)

Daniel Wigdor and Dennis Wixon. *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. Morgan Kaufmann, San Francisco, April 2011. ISBN 9780123822314. (Cited on pages 38, 42, 44, 154, and 170.)

Karl D.D. Willis, Ivan Poupyrev, Scott E. Hudson, and Moshe Mahler. Sidebyside: Ad-hoc multi-user interaction with handheld projectors. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 431–440, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. (Cited on page 115.)

Andrew Wilson, Hrvoje Benko, Shahram Izadi, and Otmar Hilliges. Steerable augmented reality with the beamatron. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 413–422, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1580-7. (Cited on page 115.)

Andrew D. Wilson and Hrvoje Benko. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 273–282, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0271-5. (Cited on page 153.)

Anna Wong, Nadine Marcus, Paul Ayres, Lee Smith, Graham A. Cooper, Fred Paas, and John Sweller. Instructional animations can be superior to statics when learning human motor skills. *Computers in Human Behavior*, 25(2):339 – 347, 2009. ISSN 0747-5632. Including the Special Issue: State of the Art Research into Cognitive Load Theory. (Cited on page 45.)

Rayoung Yang and Mark W. Newman. Learning from a learning thermostat: Lessons for intelligent systems for the home. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 93–102, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1770-2. (Cited on pages 14, 20, 26, 42, 47, 174, and 175.)

G. Michael Youngblood, Diane J. Cook, and Lawrence B. Holder. Managing adaptive versatile environments. *Pervasive Mob. Comput.*, 1(4):373–403, December 2005. ISSN 1574-1192. (Cited on page 19.)

Jamie Zigelbaum, Michael S. Horn, Orit Shaer, and Robert J. K. Jacob. The tangible video editor: collaborative video editing with active tokens. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, TEI '07, pages 43–46, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-619-6. (Cited on page 95.)

Jamie Zigelbaum, Angela Chang, James Gouldstone, Joshua Jen Monzen, and Hiroshi Ishii. SpeakCup: simplicity, BABL, and shape change. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pages 145–146, 2008. ISBN 978-1-60558-004-3. (Cited on page 95.)

Jamie B Zigelbaum. Mending fractured spaces: external legibility and seamlessness in interface design. Master's thesis, Massachusetts Institute of Technology, 2008. (Cited on page 171.)