Institut für Informatik
der Technischen Universität München

Continuous, seamless integration of users into the software design of interactive systems

Patrick Nepper

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Alexander Pretschner
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Dr. h.c. Manfred Broy
2.	Univ.-Prof. Dr. Albrecht Schmidt, Universität Stuttgart

Die Dissertation wurde am 16.05.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 18.10.2012 angenommen.

Kurzfassung

Interaktive Dialogsysteme sind interaktive Systeme, die mit menschlichen Nutzern interagieren. Es ist daher wünschenswert Nutzer in die Anforderungserhebung und -validierung solcher Systeme zu integrieren. Obwohl hierfür Methoden existieren, werden Nutzer de facto entweder gar nicht oder nur zu bestimmten Projektmeilensteinen integriert.

Eine Methode der Nutzerintegration im Rahmen der Anforderungserhebung ist die Prototypisierung von Benutzerschnittstellen. Die vorliegende Arbeit untersucht Eigenschaften dieser Methode und ihrer Artefakte, den Entwurfsprototypen. Die Ergebnisse einer explorativen Fallstudie, die im Rahmen der Arbeit durchgeführt wurde, deuten darauf hin, dass Entwurfsprototypen implizite Anforderungen enthalten, welche aufgrund ihres informellen Charakters verloren gehen können. Um den Verlust zu vermeiden ist ein erheblicher manueller Dokumentationsaufwand nötig. Dieser Aufwand steht im Widerspruch zu einer effizienten kontinuierlichen Integration von Nutzern.

Die vorliegende Arbeit führt eine Referenzarchitektur für Entwurfsprototypen ein, die eine strukturierte Repräsentation informeller Entwurfsprototypen ermöglicht. Die Grundlage bildet ein virtuelles Protokollmodel, das aus deskriptiven Theorien zu strukturellen Eigenschaften der Mensch-Maschine Kommunikation abgeleitet wird.

Um die Vorteile einer strukturierten Repräsentation zu verdeutlichen, wird ein Formalismus zur Darstellung eines spezifischen Anforderungstyps, den Entwurfsmustern für Benutzerschnittstellen, eingeführt. Eine bestätigende Fallstudie belegt, dass der vorgestellte Ansatz eine Erkennung von Entwurfsmustern in Entwurfsprototypen ermöglicht.

Die vorliegende Arbeit führt darüber hinaus einen Arbeitsablauf zur kontinuierlichen Nutzerintegration in der Anforderungsanalyse ein. Hierbei werden Nutzern Werkzeuge zur Verfügung gestellt, die eine Teilnahme ohne Expertenwissen ermöglichen. Die strukturierte Repräsentation von Entwurfsprototypen erlaubt dabei eine kontinuierliche Analyse der Ergebnisse der Nutzerintegration.

Der Arbeitsablauf und die zugehörige Werkzeugunterstützung wurden im Rahmen einer vergleichenden Fallstudie auf ein professionelles Software Entwicklungsprojekt angewendet. Die Ergebnisse der Studie deuten darauf hin, dass der vorgeschlagene Ansatz die Häufigkeit der Integration von Nutzern während der Anforderungsanalyse, die nahtlose Integration von Entwurfsprototypen und die kontinuierliche Nachverfolgung von Anforderungen verbessert. Die Ergebnisse dieser Arbeit liefern Anknüpfungspunkte für weitere Forschungsarbeiten im Hinblick auf eine Integration des vorgeschlagenen Ansatzes mit weiteren Methoden der Anforderungsanalyse.

Abstract

Interactive dialogue systems are interactive systems that are capable of interacting with human users. It is therefore desirable to integrate users for the elicitation and validation of requirements for such systems. Although methods for user integration exist, in current practice they are integrated at specific project milestones at most.

A popular method for integrating users during the requirements elicitation stage is user interface prototyping. This work examines the characteristics of this method and of the resulting design prototypes. The results of an exploratory case study conducted as part of this research suggest that design prototypes contain implicit requirements that are easily lost due to their informal nature. Avoiding a loss of requirements causes a significant effort for manual documentation. These efforts are in conflict with an efficient, continuous integration of users into the process.

This work introduces a layered reference architecture for design prototypes that allows a structured representation of informal design prototypes. It is based on a virtual protocol model, which is deduced from descriptive theories of the structural properties of human-computer interaction.

In order to show the potential benefit of a structured representation for design prototypes, a formalism is introduced for representing a specific type of requirements, user interface design patterns. A confirmatory case study confirms that this approach allows the automatic detection of user interface design patterns in design prototypes.

Furthermore, this work introduces a workflow for the continuous integration of users into requirements engineering. This is achieved by providing users with the necessary toolkit to take part without requiring expert knowledge. The structured representation for design prototypes plays a key role in this as it allows analysing the results of integrating users continuously.

The workflow and its tool support have been applied to a professional software engineering project as part of a comparative case study. The results indicate that the approach improves the frequency of integrating users into the requirements engineering phase and that it yields a more seamless integration of design prototypes and a more continuous tracing of requirements. The results of this work provide starting points for future work towards the integration of the suggested approach with other methods of requirements engineering.

Acknowledgement

First and foremost, I'd like to thank Prof. Manfred Broy for supporting my research and teaching at the Center for Digital Technology and Management (CDTM) and at his Chair for Software and Systems Engineering. It was a truly engaging environment to work in. I also like to thank Prof. Albrecht Schmidt for providing advise as the co-supervisor for this thesis and for HCI in general.

I'd like to thank Anselm Bauer (Stylight GmbH), Philipp Huy (Pidoco GmbH), and their teams for supporting this thesis as case study partners, interviewees, and by providing access to their tools and services. Also, thank you to a class of CDTM students for contributing to an empirical study as part of my graduate course on "Advanced Topics in User Interface Design". I am especially grateful for all the great students who decided to let me participate in their bachelor's, master's or project thesis: Anselm Bauer, Tilman Dingler, Felix Frank, Michael Gross, Eva Reinstadler, Andreas Schmid, Florian Sauter, Christian Schraml, Nino Ulsamer, Thilo Weghorn. Thanks also to Tilman Dingler for all the discussions and his contributions to the before mentioned graduate course.

I am very grateful for all the research advise I received from my colleagues at the chair, especially Sebastian Winter, whom I thank for all his valuable thoughts, challenging questions and his continued support for my work. I'd also like to thank Florian Deißenböck, Benjamin Hummel, Elmar Jürgens, Silke Müller, and Stefan Wagner for sharing their experience and insights. I'd like to thank my former colleagues in the CDTM management team, Ana Balevic, Mark Bilandzic, Fabian Dany, Isabel Dörfler, Rebecca Ermecke, Bernhard Kirchmair, Nikolaus Konrad, Marie-Luise Lorenz, Philip Mayrhofer, Christian Menkens, Benedikt Römer, Uwe Sandner, Andreas Schmid, Julian Sußmann, and Uta Weber, for the Center Spirit. A special thanks goes to Dennis Wetzig for being the best co-founder I can imagine.

Thank you Markus Schuster and Florian Büttner for giving me plenty of opportunities for getting distracted (and learn about physics at the same time), and for your assistance during the time of writing this thesis.

I'd like to thank my mother and my parents in law for all the emotional support. The past months have been a challenging time – especially for the people who live closest to me.

Thank you Hanna for all your support, your positivism, your incredible patience. You make my life wonderful – every day.

Contents

> The cost of fixing or reworking software is much smaller in the earlier phases of the software life cycle than in the later phases.
>
> *Barry W. Boehm*

1 Introduction

Integrating users into the software design of interactive dialogue systems (IDSs[1]) contributes to the overall product development goal [14] of designing the *right* product and designing the product *right*. Users should even be integrated *continuously* [20] during the elicitation and validation of requirements. Since users are usually not fully aware of their needs and state requirements informally [80], Design Prototypes can help to elicit and validate their requirements. The overall quality of the product with respect to its requirements is assumed to increase with the number of Design Prototypes that are created in early stages of the development process [30, 33].

1.1 Problem Statement

A recent study conducted as part of a research project at the Chair for Software and Systems Engineering suggests that customer feedback is mostly gathered at specific milestones [88]. An earlier study that has been conducted with focus on the Swiss software engineering market indicates that this holds also for users in general as in most software projects users are not integrated when it comes to requirements elicitation and UI Design [87]. This work examines reasons and possible solutions for the insufficient integration of users into the software design of IDSs.

Dix [22] argues that the design of IDSs gives rise to a *"formality gap"* between the informal requirements in the heads of stakeholders such as customers or users and a formal statement about requirements. Findings from exploratory research (see Chapters 4 and 6) support this hypothesis and suggest that the following problems contribute to the lack in continuous integration of users:

Implicit requirements. Currently, there is a gap between the *artefacts* of user-related activities and those of the requirements engineering (RE) process: Design Prototypes define user-related requirements implicitly using visual artefacts as opposed to a formal requirements specification.

Unstructured representation of the interaction. The mapping between Design Prototypes and a structured representation of the interaction between a system and

[1] The terminology used in this work is introduced in Chapters 2 and 3.

its user is unclear. This leads to a *limited traceability* of the requirements that have been elicited using such artefacts.

High manual effort needed for analysing requirements. Design Prototypes are not self explanatory and their implicit requirements are hard to understand without explicit documentation. Current methods for user integration depend on manual tasks that are not easily scalable in terms of the frequency of their execution, the number of users involved (e.g. to reflect different user segments, regions, and languages) and the number of requirements involved. Finally, no sufficient tool support for involving users seamlessly and continuously into the process has been developed to date.

1.2 Research Questions

As part of this work the following research questions (RQs) have been deduced. The research questions are refined and examined in three case studies.

Research Question	**Examined in**
RQ 1 *What are advantages and limitations of using Design Prototypes for eliciting and validating requirements?*	Chapter 4 Study partners: CDTM, Munich and Pidoco GmbH, Berlin
RQ 2 *How can realistic Design Prototypes and their implicit requirements be represented in a more structured way?*	Chapter 6 Study partner: Pidoco GmbH, Berlin
RQ 3 *How can Design Prototypes be used more efficiently for eliciting and validating requirements?*	Chapter 9 Study partners: Pidoco GmbH, Berlin and Stylight GmbH, Munich

Table 1.1: Overview of research questions and their coverage in the case studies in Chapters 4, 6, and 9.

1.3 Scientific Approach

This work addresses the research questions using the following scientific approach.

RQ 1 What are advantages and limitations of using Design Prototypes for eliciting and validating requirements?

Goal	Method	Description
Exploration	Participation, literature research	Theoretical orientation and practical involvement by conducting projects in the field of UI Prototyping
Preliminary hypotheses	Exploratory case study	Academic case study on issues for involving users in the design of IDSs

Table 1.2: Scientific Approach for **RQ 1**.

RQ 2 How can realistic Design Prototypes and their implicit requirements be represented in a more structured way?

Goal	Method	Description
Conceptual model for HCI and IDS design	Literature research, logical deduction	State-of-the-art analysis of theories for HCI and the design of IDSs
Theoretical concept	Logical deduction	Development of a theoretical model of the structure of IDSs and a structured representation of Design Prototypes and requirements
Confirmation of hypotheses	Confirmatory case study	Confirmation of preliminary hypotheses by examining the properties of realistic Design Prototypes

Table 1.3: Scientific Approach for **RQ 2**.

RQ 3 How can Design Prototypes be used more efficiently for eliciting and validating requirements?

Goal	Method	Description
Theoretical concept	Logical deduction	Development of a concept for an improved workflow and tool support based on the theoretical concepts and study findings
Functional prototype	Design research, prototyping	Creation of a prototypical tool support according to the preceding results as a basis for a subsequent validation
Validation	Comparative case study	Comparative case study with an industry partner in order to compare the improved process with the state-of-the-art

Table 1.4: Scientific Approach for **RQ 3**.

The scientific approach includes empirical case studies that have influenced the theory building process and provide some validation of the contributions:

- The first, explorative case study results in the preliminary hypothesis for this work: Design Prototypes are informal representations of numerous requirements across different layers of HCI.
- The second case study confirms the preliminary hypothesis by identifying UI Design Patterns in 49 Design Prototypes from an external study partner based on a mapping of Design Prototypes and UI Design Patterns to the structured representation that is introduced in this work.
- The third case study provides some validation for the contributions of this work by applying the suggested workflow and tool-support to a professional software engineering project.

The case studies presented in this work are structured according to the case study research process outlined in Runeson and Höst 2009 [70]. The only exception is the discussion of related work which is presented separately in Chapters 2 and 3.

1.4 Contribution

This work contributes to model-driven development [11] of interactive dialogue systems, by extending the overall framework for requirements engineering [12]. Following the scientific approach outlined above, this work aims at the continuous, seamless integration of users into the software design of interactive dialogue systems with the following contributions:

- a literature review of structural approaches to modelling HCI
- a conceptual model of Human-Computer Interaction for IDSs
- a reference architecture for Design Prototypes of IDSs and its formalization with FOCUS [13]
- a structured representation for UI Design Patterns
- a workflow for integrating users continuously and seamlessly
- prototypical tool support
- answers to the research questions and validation of this work's contributions based on three case studies

Figure 1.1 illustrates the contribution and the interrelation of its elements.

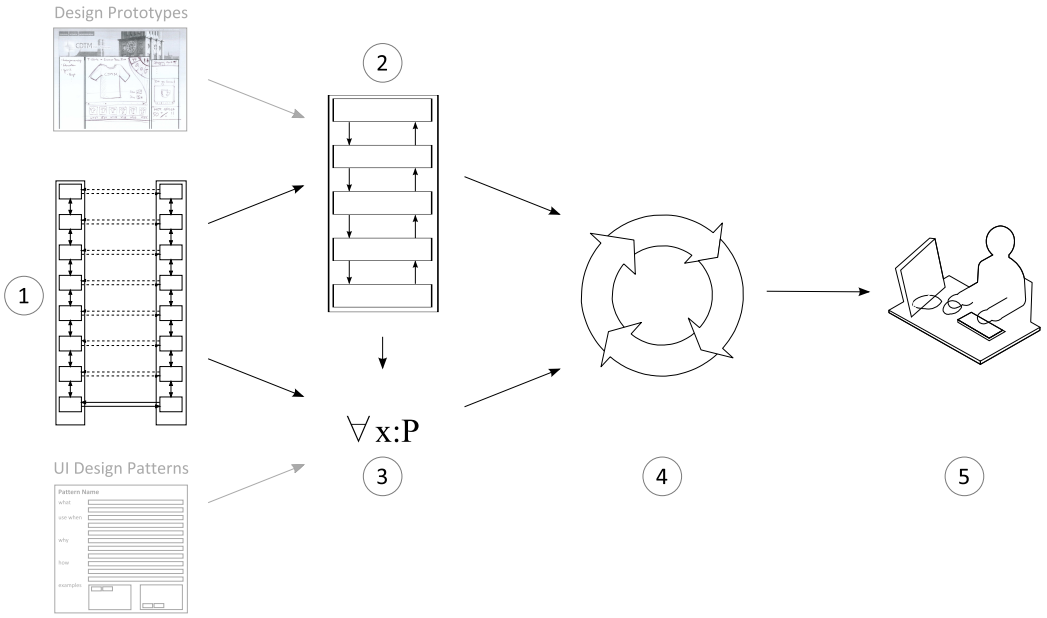

Figure 1.1: This work contributes (1) a conceptual model of HCI for IDSs, (2) a reference architecture for Design Prototypes, (3) a structured representation for UI Design Patterns, (4) a workflow for continuous user integration, and (5) tool support and three case study results (not depicted).

1.5 Contents

The remainder of this thesis is organized as follows: Chapters 2 and 3 introduce the relevant aspects of Human-Computer Interaction and the design of IDSs as well as the terminology used. Chapter 4 presents the results of an exploratory case study, which motivates the preliminary hypotheses as presented above. Chapter 5 introduces a reference architecture for Design Prototypes and a structured representation of UI Design Patterns. It is used for analysing implicit requirements in Design Prototypes from professional software projects as part of a confirmatory case study in Chapter 6. The reference architecture for Design Prototypes is also the theoretical basis for introducing a workflow for integrating users into the design of IDSs in Chapter 7 and the respective tool support in Chapter 8. The validation of the workflow and its tool support in a professional environment is discussed in Chapter 9. Chapter 10 provides a discussion of the main findings of this work as well as an outlook on the most prominent questions that provide starting points for future work towards continuous and seamless integration of users into the design of IDSs.

> Almost all techniques for improving usability do not refer to the device itself; they have no theory of how interaction with the device works.
>
> *Harold Thimbleby*

2 Interactive Dialogue Systems

The inherent structure of *Human-Computer Interaction* (HCI) needs to be understood in order to be able to design *Interactive Dialogue Systems* (IDSs) that are efficient and effective to use, and to identify and resolve shortcomings in their design. This chapter introduces the relevant terminology for this work and a literature review of the state-of-the-art of related theoretical models. Based on this summary a theoretical model for the structure of HCI is deduced and applied to examples of interaction. This theoretical model is the basis for the definition of a reference architecture for Design Prototypes in Chapter 5.

2.1 Terminology

The field of HCI is influenced by various fields of research and therefore most of its terminology is used with different notions in different contexts. The following section presents an overview of the terminology as it applies to the remainder of this work.

Human-Computer Interaction[1] (HCI). The discipline of Human-Computer Interaction is concerned with "the *design*, *evaluation* and *implementation* of interactive computing systems for human use and with the study of major phenomena surrounding them" [37]. In this work, the term HCI will be used specifically to denote the *interaction* between human users and IDSs (as opposed to the *discipline*).

Interactive System (IS). In general, IS denotes a system, which "consist[s] of a set of components cooperating and exchanging information through interaction" [13]. The components of an IS are called *interactive components*. One of several possibilities to categorize these components is according to the application domains for which they are used, e.g. process control, telecommunication networks, HCI, etc.

User Interface (UI). The *UI* of an IS consists of "those aspects of the system that the user comes in contact with – physically, perceptually, or conceptually." [52]

User Interface Toolkit (UI Toolkit). Although this term is frequently used in publications, no commonly cited definition exists. In the following, the term *UI*

[1] Frequently used synonyms: Man-Machine Interaction, Computer-Human Interaction.

Toolkit denotes a software library that facilitates modality-specific interaction between a physical input or output device and an IS. Typical examples are *Graphical UI (GUI)*, *Voice UI (VUI)* and *Tangible UI (TUI)* toolkits.

Interactive Dialogue System (IDS). This work focuses on ISs which contain interactive components that are capable of interacting with a human user. In this work, an IS for the domain of HCI will be denoted as an *Interactive Dialogue System*. Fraser coined the term *dialogue system* in 1997 as a computing system "with which humans interact on a turn-by-turn basis" [28]. Since then HCI has evolved towards multimodal (e.g. GUI +TUI), mixed-initiative interaction with multiple simultaneous conversations, and potentially including implicit interactions [74]. In the following the term *dialogue* will be used in its broadest sense to refer to all those interactions.

IDSs, specifically multimodal IDSs (combining modalities such as *graphical*, *voice* and *haptic* interaction), have been the subject of numerous empirical and applied research projects in the past. The results of a recent study [47] indicate, that mixed-initiative dialogues between IDS and user can reduce the overall task completion time for user tasks significantly (16.8% in the context of the study). A summary of related research projects can be found in [55]. Explorative works by the author introduce a software architecture for multimodal location-based services together with a prototypical implementation of a multimodal, mobile tourist guide [57], and a prototypical implementation of a multimodal, multimedia customer information service for brick and mortar stores [56].

2.2 Modelling Human-Computer Interaction

This section introduces theories that try to explain the fundamental structure of HCI as a starting point for building theoretical models about the structure of HCI in IDSs. Theories can be *descriptive* or *prescriptive* in nature.

Elements of *prescriptive theories* for developing IDSs such as UI Design Patterns (cf. Section 3.2.2), usability guidelines [20], reference architectures for multimodal IDSs e.g. for context-aware mulit-sensor systems [75] (cf. [55] for an overview of hub-and-spoke and agent-based architectures), design processes for IDSs [78, 16] and informal modelling notations based on UML [17, 51] are well known to HCI experts. These theories and their elements aim at the *improvement* of the discipline of HCI.

Descriptive theories, on the other hand, aim at *building knowledge* about the nature of HCI. This reflects a scientific approach to understanding phenomena of HCI as a basis for modelling reality and thus for being able to improve current approaches to designing IDSs (cf. Chapter 3). This section provides an overview of the state-of-the-art of descriptive theories for modelling HCI, which lies at the heart of every IDS. It starts off with an introduction to the different dimensions of modelling HCI, which is followed by a discussion of individual *structural* approaches.

The interaction between human users and IDSs can be analysed along different dimensions, which can be grouped into *structural* and *behavioural* aspects.

Structural aspects of the interaction:

Levels of abstraction	describe the different *layers* of interaction. Layers are essential for structuring complex interactions so that the conceptual difference of physical interaction, its interpretation and its relation to the overall conversation between IDS and user becomes evident.
Communication channels	describe the channels that allow for communication between and within the layers of interaction.
Protocols	describe the exchange of messages across communication channels.

Behavioural aspects:

Action models	describe the steps in which interaction proceeds.
Interaction coordination	describes the way in which action models are processed.

As we are interested in gathering a better understanding of the structure of Design Prototypes, this work focuses on structural aspects of HCI, which are introduced and discussed in the next section. Bass et al. [1], Nielsen [58] and Shneiderman [78] provide additional details to these and other approaches to modelling HCI.

2.2.1 Structural Approaches

Most descriptive theories with a focus on structural aspects try to separate the interaction into different layers and describe the communication channels and protocols between them. This approach to the separation of concerns is a basic concept in computer science and has been used e.g. in the design of the OSI/ISO reference model [93]. The analysis in this section

- reviews and summarizes the most relevant structural approaches to modelling HCI,
- helps in building a common terminology[2] for the different layers of interaction, and
- discusses the advantages and disadvantages of each approach with a focus on their structural aspects.

Figure 2.1 illustrates how these approaches influenced each other historically. The results of the following analysis will provide the basis for a conceptual model of HCI for IDSs in Section 2.3.

[2]The terminology borrows some established terms from the field of *computational linguistics* [48, 49], namely *alphabet, lexeme, lexicon, morpheme, syntax, semantics* and *conversation.*

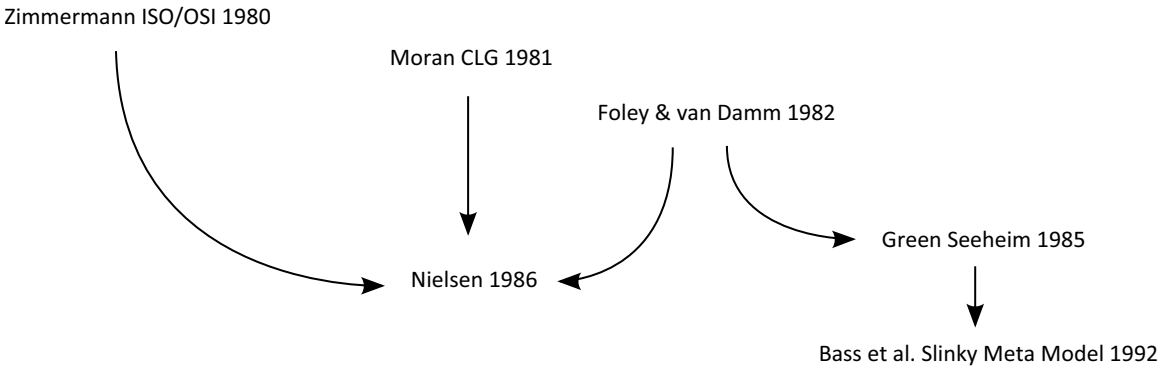

Figure 2.1: Historical overview of the structural approaches discussed in this chapter (based on mentions in [31, 27, 52, 58, 93]).

Command Language Grammar

The conceptual model of the Command Language Grammar (CLG) by Moran [52] introduces six layers (cf. Figure 2.2) to describe IDSs [3]. The *task* layer contains a representation of the user's tasks and their structure. The *semantic* layer is responsible for translating user tasks into actionable operations and vice versa. The *syntactic* layer translates between abstract operations on the semantic layer and discrete, modality-specific commands. The *interaction* layer resolves syntactic commands to physical actions and vice versa. The *spatial layout* layer describes the "arrangement of input/output devices and the display graphics". The *device* layer describes the remaining facets of the physical interaction.

Figure 2.2: The Command Language Grammar places a conversational *interaction layer* between syntactic and physical layers (based on Moran 1981 [52]).

Discussion. CLG proposes an *interaction layer* for handling an abstract representation of the physical interaction as well as the conversation management, and *spatial and device layers* for managing the physical interaction. The separation of the physical interaction into spatial and device layers is mentioned but not discussed in detail by the authors. It can be criticized as being device dependent and not a generic feature of IDSs [58]. The interaction layer contains the conversation management although it is below the syntactic, semantic and task layers. This is due to the fact that the model was not solely intended to describe IDSs, but also the design process of IDSs [52, p. 3]. In this model the interaction in terms of conversation management was to be designed after specifying the syntactic and semantic layers.

[3] Moran's work focuses on "Command Language Systems" as opposed to "Natural Language Systems". Nevertheless, his findings are of general interest to the design of IDSs and have been discussed as such by other authors [58, 15]

Foley & Van Dam Model

Human dialogue — whether in spoken or in written form — is the inspirational basis for the Language Model by Foley & van Dam [26, 27]. It distinguishes between four layers (cf. Figure 2.3): The *conceptual* layer represents the user's model of an IDS. It defines objects, their properties, relationships and the possible actions on objects from the user's perspective. The *semantic* layer defines the abstract information that can be derived from user actions and abstract information that needs to be output to the user. The *syntactic* layer defines the sequencing of input and output operations through the UI. The *lexical* layer translates between syntactic data and logical interaction elements that are derived from hardware primitives.

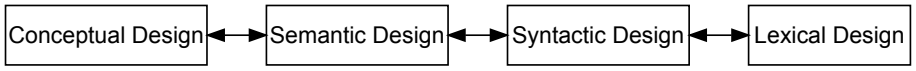

Figure 2.3: In the Foley & van Dam Model, physical interaction is part of the lexical layer (based on Foley et al. 1982, in the updated version of 1990 [27]).

Discussion. The Foley & van Dam Model defines a lexical, syntactic, semantic and conceptual layer. Concluding from the description of the different layers, the terminology does not reflect their responsibilities: The conceptual layer represents a task-level view on the interaction. The lexical layer mixes aspects of physical interaction and its abstract interpretation; the syntactic layer is not responsible for merely structuring elementary logical actions, but also for their semantic interpretation; the semantic layer takes a semantic interpretation from the syntactic layer and coordinates the current state of the conversation with the user. The model does not distinguish between physical interaction and its abstract interpretation. Buxton [15] therefore suggests adding a fifth layer (*pragmatics*) in order to provide for a better understanding of the layer that *"has one of the strongest effects on the user's perception of the system."* [15, p. 32]. The specific concept of a pragmatics layer has later been criticized by Nielsen [58, p. 307–308], who identified the need for virtual communication channels, thus reducing the conceptual importance of the physical layer. The Foley & van Dam Model does not provide deeper insight into the communication between layers [58].

Seeheim Model

During the early 1980s the term "User Interface Management System" (UIMS) was dominant in most publications on the matter of structuring HCI in the domain of graphical UIs. Developing a UIMS as a separate component in an IDS was considered to provide several advantages: Avoiding dependencies between the application and the UI and an increased usability, reliability and consistency of the UI [84]. The Seeheim Model introduced by Green in [31][4] has become one of the most cited early

[4]named after the location where the EUROGRAPHICS Workshop on UIMSs took place in 1983

contributions to modelling IDSs (see [32] for a discussion of an early implementation of an IDS based on this model).

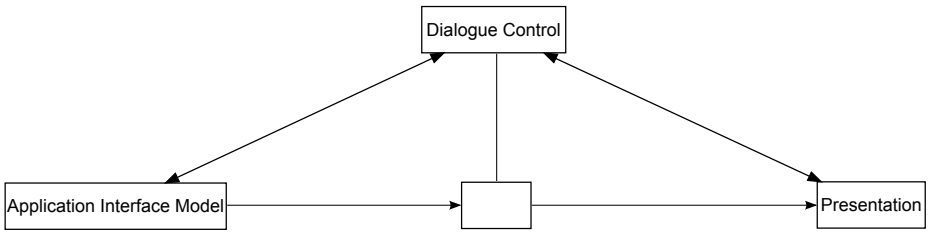

Figure 2.4: The Seeheim Model allows for direct communication between the *presentation component* and the *application interface model* (based on Green 1985 [31]).

The Seeheim Model distinguishes between three components: The *presentation* component is responsible for the physical interaction with the user and the abstract representation of input and output data. The *dialogue control* component coordinates the sequences of input and output tokens that are produced by the presentation component and the application interface model. The *application interface model* provides façades for the functions offered by the application in order to make them accessible for the dialogue control component. The Seeheim Model explicitly allows for direct communication between the presentation component and the application interface model, once a respective channel has been set up by the dialogue control component, which acts partly as a mediator between the presentation component and the application interface model (cf. Figure 2.4). The concept of a mediator distinguishes the Seeheim Model from the Model-View-Controller design pattern [42].

Discussion. The *presentation component* is a black box that manages the physical interaction and its abstract interpretation. The *dialogue control component* manages the conversation, and the *application interface model* encapsulates the tasks offered by the application logic (which is not part of the model). The missing separation of *physical interaction* and its interpretation on a *lexical, syntactic* and *semantic* level is a limiting factor for understanding the structure of HCI.

The Seeheim Model is not a layered model as it allows for direct communication between all three components. The direct communication is motivated by performance considerations (cf. [31, p. 19]), which might be relevant for prescriptive guidance for the implementation of IDSs, but does not qualify for a descriptive theory. The direct communication between the application interface model and the presentation component limits the separation of concerns. In order for the communication to work, the application interface models needs to have knowledge about modality specific formats of input and output data as well as the current state of the interaction.

Slinky Meta Model by Bass et al.

The Slinky Meta Model by Bass et al. [1] is a meta model for the design of IDSs and a generalization of the "Arch Model" [1] (see [53] for a client-server architecture based

on the Arch Model). The author's goal was to generalize upon existing models such as the Seeheim Model to reach a descriptive theory for the structure of IDSs. The Slinky Meta Model distinguishes five levels (cf. Figure 2.5): The *domain-specific* layer contains domain-specific application logic. The *dialogue* layer is responsible for managing the task-level conversation with the user. The *domain-adaptor* layer mediates between the dialogue layer and the domain-specific layer by adding façades to domain-specific functionalities which are needed to allow for structuring domain-specific functionalities into user tasks. The *presentation* layer mediates between the interaction toolkit layer and the dialogue layer by translating between modality-independent presentation objects (e.g. selection menus) and modality-specific interaction objects (e.g. GUI elements). The *interaction toolkit* is responsible for the physical interaction between a user and the IDS.

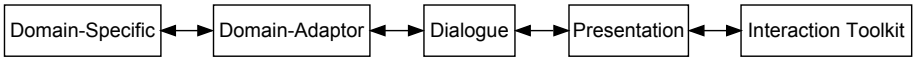

Figure 2.5: The Slinky Metamodel incorporates physical interaction into the concept of UI Toolkits (based on Bass et al. 1992 [1]).

Discussion. The Slinky Meta Model proposes an *interaction toolkit layer* for physical interaction and its abstract representation and a *presentation layer* that incorporates modality-specific syntax and semantics. The Slinky Meta Model therefore does not distinguish between the interaction that takes place on a physical level and its abstract interpretation. This is due to the fact, that the design of the Slinky Meta Model was based on the assumption that UI Toolkits (in the sense of e.g. a graphical UI Toolkit) are elementary building blocks of an IDS [1, p. 34]. This, and the missing differentiation between syntax and semantics, limits the suitability of this model for understanding a significant part of the interaction between human users and IDSs.

Virtual Protocol Model by Nielsen

The virtual protocol model by Nielsen [58] is inspired by the OSI/ISO reference model [93] and historically a successor of the other models discussed in this section. Its main contribution is the idea of virtual communication channels existing between user and IDS on each layer. Nielsen proposes seven layers (cf. Figure 2.6): The *goal layer* is where the cognitive model of the user and his real-world operational goal of the interaction reside. It is only present on the computer side for theoretical purposes as future systems might be able to set their own goals. The *task layer* contains the concepts that are part of the IDS, which themselves are representations of the real-world concepts of the goal layer. The concepts of this layer can be operationalised on the next layer, the *semantics layer*. This layer contains the functionalities that are offered by the IDS. The functionalities are the elementary building blocks for fulfilling tasks from the task layer. The *syntax layer* determines the structure and sequence of input and output tokens with respect to the respective modality being used. The *lexical* layer contains representations of the smallest meaning-carrying units of interaction, called tokens. Different tokens may carry the same information,

e.g. the word “delete” and a graphical icon with a trash symbol. The smallest information-carrying symbols, called lexemes, can be found in the *alphabetic layer*. These symbols do not have meaning themselves, but by combining different lexemes in a sequence, a meaningful token can be constructed. The *physical layer* represents physical interaction, e.g. by touch, sound, light, etc., taking place between human and computer.

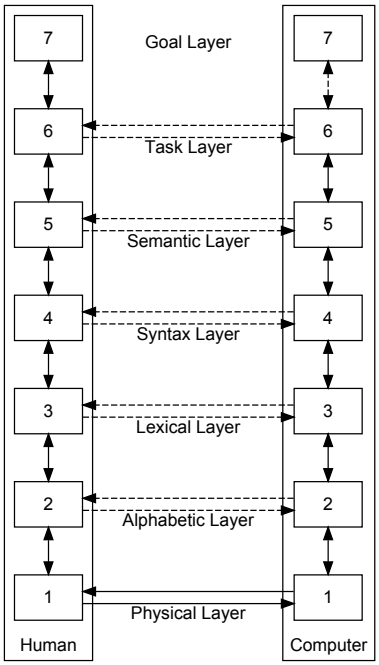

Figure 2.6: The Virtual Protocol Model applies the concept of virtual communication channels to HCI (based on Nielsen 1986 [58]).

Discussion. The lower five layers of the Virtual Protocol Model describe the structure of modality specific interaction. The *task layer* does not explain how different modalities interact and how a conversation between user and system is coordinated. The *goal layer* remains a conceptual layer that is not directly mapped to the design of an IDS. The *semantic layer* which is placed between the task layer and the syntax layer contains the functionalities of the system. The placement of the functionalities in the semantic layer instead of the task layer makes the task layer superfluous and obscures the differences between semantic interpretation of syntactic tokens and the functionalities that are offered by the system.

2.2.2 Summary

This section explored existing structural approaches to modelling HCI. To summarize the discussion, Table 2.1 offers a unified overview of all layers of interaction that have been found as part of the literature review.

Layer	Seeheim	Moran	Bass	Foley	Nielsen
Goal					
Application					
Task					
Conversation					
Semantics					
Syntax					
Lexical					
Alphabetic					
Spatial					
Physical					

contained

contained, but different order

not contained

Table 2.1: The presented approaches focus on different aspects of HCI.

Six layers of interaction are most dominant in all approaches, namely the *task*, *conversation*, *semantics*, *syntax*, *lexical*, and *physical* layers. Building on these insights and the Virtual Protocol Model by Nielsen [58], a Conversational Virtual Protocol Model for IDSs will be deduced in Section 2.3.

2.3 Conversational Virtual Protocol Model for Interactive Dialogue Systems

The Virtual Protocol Model by Nielsen is the most comprehensive descriptive, structural theory for HCI compared to the other models reviewed in the last section. In this work, a theoretical model is needed that specifically reflects HCI for IDSs. The focus on IDSs introduces some additional requirements that are not reflected by the Virtual Protocol Model as is:

Application Logic The UI is an important, but not the only part of an IDS. Therefore, the task layer of the UI model needs to be connected to the application layer, which handles the domain-specific execution of user tasks.

Conversations A conversation layer is a concept in its own right in that it is different from both task and semantic layers. The conversation layer keeps track of the turn-by-turn information exchange between user and IDS towards completing a certain task (or a part thereof). The task layer is responsible for structuring user tasks and to make them actionable by mapping user tasks to system functionalities from the application layer. However, the conversation layer is responsible for eliciting the necessary information from the user that is needed to trigger a user task from the task layer. This differentiation is important, because *a)* the structure of the turn-by-turn conversation between user and system may be

different from the structure of the user tasks and *b)* different approaches to structuring the conversation can be used to support the same user tasks, including the use of different modalities.

User Goals The goal layer of Nielsen's model is necessary for understanding the user's motivation for interacting with an IDS. It is the cause for interaction, but - as Nielsen clarifies - it is not part of an IDS as goal setting is solely a human cognitive ability.

Due to these specifics of IDSs, this section introduces a *Conversational Virtual Protocol Model for IDSs*. It is based on the Virtual Protocol Model by Nielsen [58] and the analysis from section 2.2. The proposed model extends the seven layers of the Nielsen Model with a layer for conversation management and a layer for application logic, and removes the goal layer (cf. Figure 2.7).

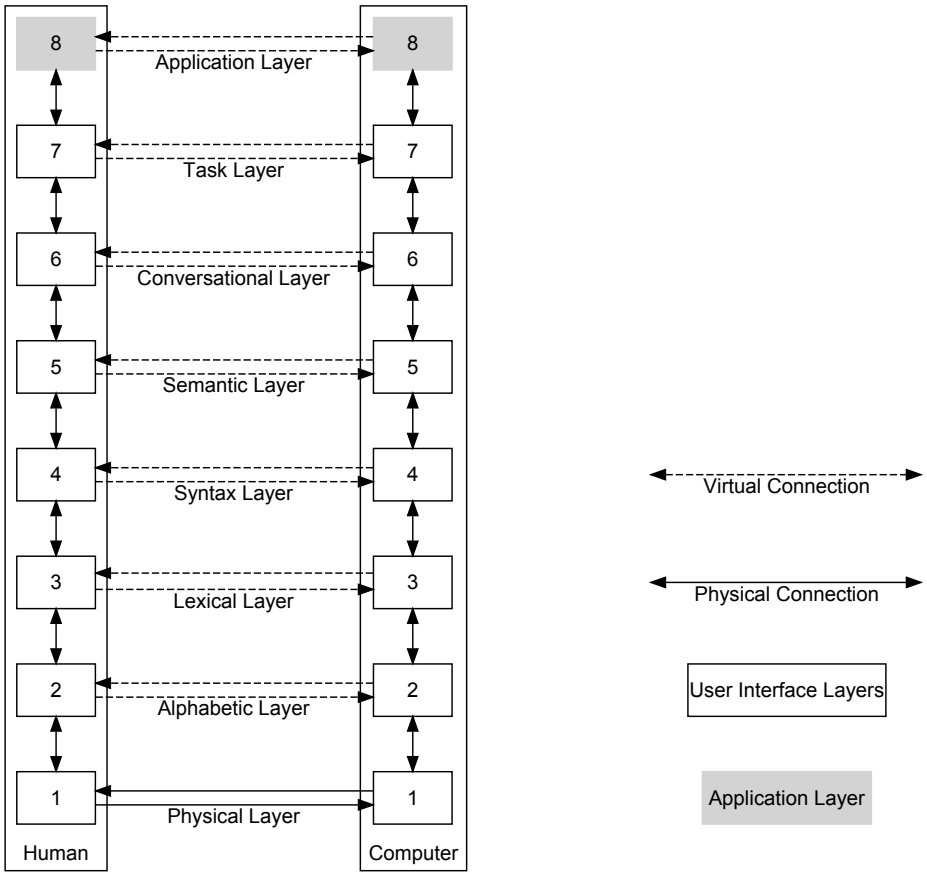

Figure 2.7: The suggested Conversational Virtual Protocol Model extends the Nielsen Model [58] with a *conversation layer* and an *application layer*, and removes the *goal layer*.

Application Layer.

The application layer contains the business logic of the IDS. The application layer is completely independent from the UI layers and has no knowledge about the UI.

The application layer provides interfaces to functions that are available for use by the UI.

2.3.1 User Interface Layers

The following provides a more detailed description of the seven user interface layers of the proposed Conversational Virtual Protocol Model for IDSs.

Physical Layer. The physical layer handles the physical interaction between user and system. This is where e.g. acoustic sounds are transmitted, screens are touched, light is emitted by LEDs and so forth. The physical layer translates between physical signals and digitized signals.

Alphabetic Layer. The alphabetic layer translates between digitized signals and abstract tokens that represent the smallest information-carrying elements of the physical interaction, e.g. between sound samples and individual letters, between coordinates of a physical touch on a sensor device and pixels of a screen position or between LED light values and colours of pixels on a screen.

Lexical Layer. The lexical layer is responsible for translating between information-carrying tokens from the alphabetic layer and meaning-carrying tokens for the syntax layer. This is what usually happens at the interface of device drivers and UI Toolkit: A physical click is translated into a mouse click event and a written word is translated into a sequence of acoustic sounds to be generated by an output device.

Syntax Layer. The syntax layer translates between sequences of meaning-carrying tokens from the lexical layer and well-formed expressions representing logical actions for the semantic layer. This is what typically happens inside of a UI Toolkit: sequences of e.g. mouse clicks and moves are translated into a drag-and-drop operation from a specific start position on the screen to a specific destination on the screen.

Semantic Layer. The semantic layer is responsible for translating between logical actions from the syntax layer and a semantic interpretation for the conversational layer, e.g. translating a sequence of keyboard press events into a text input event, or translating a mouse point-and-click action into a button push event.

Conversational Layer. The conversational layer manages the turn-by-turn conversation between human user and IDS. It knows about the current state of the conversation, i.e. the goal of the conversation (which is the completion of a user task), the steps in which the goal can be reached, the type of information necessary to complete the current step and proceed to the next step, and the information already gathered. This layer contains the necessary conversational algorithms that decide how to gather information from the user, which is not part of the structural model, but an important aspect of the behavioural model. The conversation layer translates between the semantic interpretation from the semantic layer and task-level operations from the task layer.

Task Layer. The task layer is responsible for wrapping the business logic of the application layer into actionable user tasks. It is therefore the main interface of the UI towards the core application (in other terms of the front-end to the back-end). The task layer contains definitions of user tasks that are supported by the IDS and their requirements e.g. in terms of information that needs to be gathered for them to be actionable. The task layer translates between task-level operations visible to the conversational layer and sequences of function calls available from the application layer.

Table 2.2 informally illustrates examples of interactions between IDSs and humans through different modalities and their mapping to the eight layers.

2.3.2 Summary

The *Conversational Virtual Protocol Model* proposed in this section extends Nielsen's Virtual Protocol Model with *a)* a conversational layer that reflects the turn-by-turn interaction between user and system towards fulfilling a user task and *b)* an application layer that reflects those parts of an IDS that are not part of the UI. This model will be necessary to define a reference architecture for Design Prototypes and a mapping of otherwise ambiguous elements of prescriptive theories such as UI Design Patterns to a structured representation in Chapter 5.

Layer	Type of Information (ToI)	Example
Application Layer	Function/Service	Application software implementing the task “bookHotel”
Task Layer	Operation	bookHotel(id, date, price)
Conversation Layer	Dialogue state and structure	dialogue[action=booking, state=stateObject]
Semantics Layer	Semantic interpretation	statement[type=confirmation, value=ok]

	VUI		GUI		Haptic UI	
	ToI	Example	ToI	Example	ToI	Example
Syntax Layer	Sentences	“Yes, please.”	UI element event	OkButton. click	UI element event	OkButton. click
Lexical Layer	Word	“Yes”	Click event	Mouse click event: left button at screen position x,y	Touch event	Point touch event at screen position x,y
Alphabetic Layer	Morpheme	[yɛ]	Input device event	<left button press event>	Input device event	<coordinates of touch field on touch-screen>
Physical Layer	Sound wave	<sound sample>	Pushing an input device	<pushing left button of a mouse>	Physical touch	<touch with right index finger on touch-screen>

Table 2.2: Illustrative examples of mapping different modalities of HCI to the layers of the Conversational Virtual Protocol Model.

> A prototype is a prototype, regardless of the technology that is used to implement it, and regardless of its fidelity relative to the actual product.
>
> *William Buxton*

3 Designing Interactive Dialogue Systems

Chapter 2 introduced *descriptive* theories for the interaction between human users and IDSs. This chapter will provide an introduction to elements of *prescriptive* theories, which are relevant for the day-to-day work of professionals in the discipline of HCI, specifically for those involved in the design of IDSs.

The UI is the prominent part of an IDS through which the design of the system becomes evident to its user. It becomes a key element during the *elicitation* of requirements as it reflects design decisions across all conceptual layers of an IDS (cf. Chapter 2). The following segments discuss the state-of-the-art of gathering requirements for IDSs through *prototyping* their respective UIs and making use of UI Design Patterns.

3.1 Terminology

This chapter touches terminology from a field of research that has gone through various phases of nomenclature with *User Interface Design, Usability Engineering, Participatory Design, User Centered Design, Interaction Design* and *User Experience* being the most dominant terms. These names and their changing use reflects the changing focus and evolution of the discipline of designing IDSs over time. This section provides a definition of terms that will be used in the remaining chapters.

Requirements Engineering (RE). Sommerville distinguishes between agile and plan-driven software engineering processes, which can be based on different process models, namely the waterfall model, incremental development or reuse-oriented software engineering [80]. Irrespective of the process and model used, all software engineering processes involve activities for *specification*, *development*, *validation* and *evolution* [80]. This work focuses on the sub-process that is necessary for the activity of software specification: *Requirements Engineering* (RE). In the following, RE will refer to the RE process, which is considered to consist of *requirements elicitation, analysis, validation and management.*

User Interface Design (UI Design). In this work the term UI Design will be used to denote the discipline that is concerned with the *design* of the UI of IDSs. In this

respect, *design* refers to the process of elicitation, specification and validation of UI-related *requirements*. The existing approaches to UI Design can be separated into two different categories: *Model-based UI Design* and *UI Prototyping*. Both approaches serve different purposes (see below). Therefore, they are not necessarily mutually exclusive, but can be applied in different phases of a UI Design process.

User. The term *user* denotes the role of a human that is interacting with an IDS. Users can be categorized by their *personal properties* [65] (e.g. experience, attitudes, occupation, etc.) *and* their *role* in the UI Design process (e.g. internal test user, external test user, customer, end user, etc.). In this work the term *user* will be used in its general sense and will be narrowed down with respect to specific categories where necessary.

User Centered Design (UCD) Process. A UCD process is characterized by an iterative process, active user involvement, and a proper understanding of user and task requirements [25]. Typical activities that are involved with a UCD process are so called *Usability Activities* [25].

Usability Activities. According to ISO 13407 Usability Activities comprise: Analysis of the context of use, specification of user requirements and organizational requirements, UI Design, and the evaluation of the UI Design against the requirements [50].

Model-based UI Design. In a model-based approach to UI Design [90], both the UI as well as related requirements are defined using formal models. This allows the application of formal software engineering methods and tools to UI Design, e.g. the verification of requirements.

UI Prototyping. Instead of applying formal methods, UI Prototyping [14] is an approach to UI Design that is based on the (mostly iterative) creation of Design Prototypes. This allows the application of Usability Activities, e.g. for the elicitation and validation of requirements.

Design Prototype. In order to distinguish between functional prototypes, which can be found in later stages of a software engineering process, prototypes created during the RE phase of IDSs will be referred to as *Design Prototypes* in the remainder of this work.

Evaluation Activities. The term evaluation is used with different notions by UI Design experts and software engineers. In this work the term *Evaluation Activities* will be used as a hypernym for *Verification Activities* and *Validation Activities*. *Verification Activities* aim at verifying that a given Design Prototype fulfils a given set of requirements. *Validation Activities* aim at validating requirements with users, i.e. examining whether a given set of requirements reflects actual user requirements.

User Toolkit. In order to be distinguishable from the term *UI Toolkit* introduced in Chapter 2, the term *User Toolkit* is used as a shorthand term for *toolkits for user innovation and design*, which "are integrated sets of product design, prototyping, and design-testing tools intended for use by end users. The goal of a toolkit is to enable non-specialist users to design high-quality, producible custom products that exactly meet their needs." [38, p. 147]

Collaborative UI Design (Collaborative UI Design). A *collaborative* approach to UI Design is characterized by incorporating those Usability Activities which are best suited for involving users and other stakeholders during the design of IDSs, i.e. [20]: discussion of drafts of a concept for user interaction, validation of user requirements, discussion of prototypes, iterative usability evaluation and iterative usability testing with the integrated system. User Toolkits facilitate Collaborative UI Design by providing non-specialist stakeholders such as internal and external test users, colleagues from other disciplines and external partners with the means to contribute to the design process.

3.2 Design Prototypes

Design Prototypes exist for different modalities of interaction with very different levels of fidelity, based on different sorts of material, created with different types of tools. Design Prototypes often are extreme simplifications of the system for which they are created: A paper mock-up may be a Design Prototype of a GUI-based system, a wooden prototype may be a Design Prototype of a car, and a person sitting behind a computer display may be part of a "Wizard-of-Oz" [59] prototype of a VUI-based system. Still, Design Prototypes *are* prototypes, because they are used to reflect certain aspects (such as certain functionality, visual design, the UI, etc.) of a system or parts thereof before its final implementation. This is important to note, as these prototypes need to obey requirements just like any other type of prototype, if they are to be used in professional software engineering projects.

However, Design Prototypes are used to serve a different purpose than functional prototypes: Design Prototypes are used for the elicitation of requirements and early validation of these requirements with customers, users, software engineers and other stakeholders in the software engineering process (cf. Section 3.3.1). With respect to the design of IDSs, Design Prototypes are prototypes that are created as part of the UI Design.

3.2.1 Design Prototypes for Graphical User Interfaces

The following segments introduce two specific types of Design Prototypes: The *Paper Prototype*, which is an analogue, paper-based prototype of the UI of an IDS, and the *Electronic Prototype*, which is an electronic, software-based representation. They have two properties in common: *a)* they are used for prototyping *Graphical* UIs, and *b)* they belong to the most frequently used Design Prototypes in software engineering. The use of both types for Usability Activities will be discussed in Section 3.3.1.

Paper Prototypes

A Paper Prototype is a paper-based representation of a GUI. It consists of one or more paper mock-ups and one or more storyboards (cf. Figure 3.1). A *paper mock-up* is a visualization of the visible parts of a GUI in a specific state.

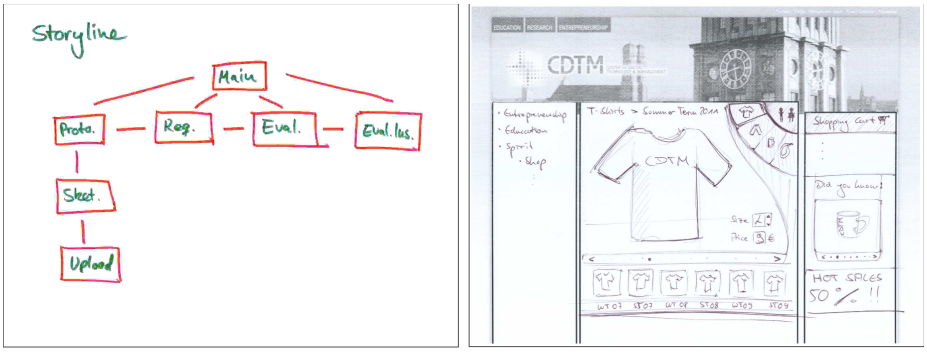

Figure 3.1: A storyboard (left) aligns mock-ups into an interaction scheme. A paper mock-up (right) visualizes the visible parts of a GUI (created during the exploratory study in Chapter 6).

A *Storyboard* can be used to align the different mock-ups into an interaction scheme, which reflects the conversational steps the interaction between the user and the system should follow in a certain use case.

Electronic Prototypes

Instead of mock-ups, Electronic Prototypes consist of *wireframes*. A wireframe is similar to a mock-up in that it represents the visible parts of a GUI. The key difference is a second key element: *Links* explicitly connect two wireframes by defining a relation between GUI elements (such as buttons) and (target) wireframes (cf. Figure 3.2).

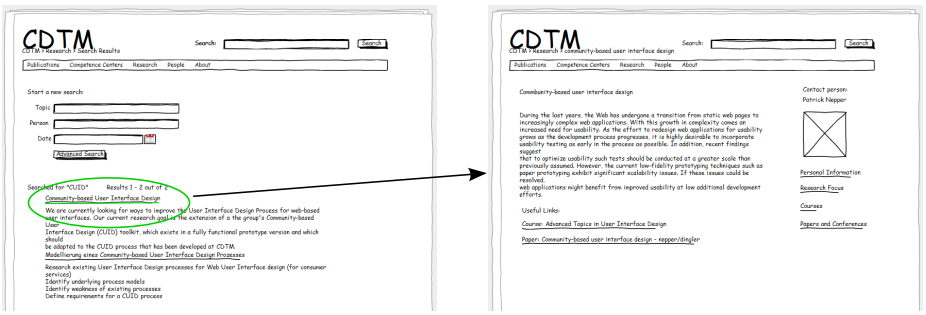

Figure 3.2: Wireframes are similar to mock-ups and contain *links* (highlighting and arrow added for illustration).

A *Storyboard* is used to group wireframes into use cases, which hides some of the complexities of all the underlying links (cf. Figure 3.3).

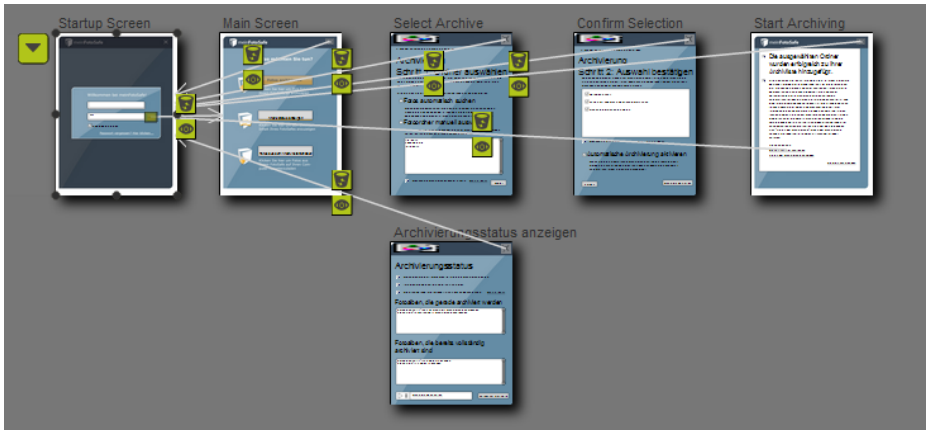

Figure 3.3: A Storyboard visualizes the links between wireframes (Source: Pidoco GmbH, Berlin/AllMyShots GmbH, Munich).

3.2.2 UI Design Patterns

Design Patterns [29] provide guidance to software and systems design by collecting and documenting best practices. The Design Patterns for the composition of software systems ("structural patterns") or solutions to basic algorithmic problems ("behavioural patterns") in [29] are defined using the notations of the Object Modelling Technique [69] and Interaction Diagrams [9]. The concept of Design Patterns has been transferred to UI Design as well. Design Patterns have been documented for UIs of different modalities, e.g. graphical UI Design Patterns [85] and voice UI Design Patterns [76]. The following section provides a closer look at UI Design Patterns for GUIs.

Tidwell [85] introduces more than 80 UI Design Patterns which are categorized by problem areas. Each pattern consists of a *name*, a short abstract (*"what"*), descriptions of *when, why* and *how* to apply it, and illustrative *examples*. Figure 3.4 provides a schematic depiction of the structure of UI Design Patterns. The *what* and *how* segment of a UI Design Pattern describe the properties of the pattern with respect to its visible elements and interaction schemes. The description itself is prosaic and requires human interpretation in order to be applied to a specific IDS. Specific patterns will be discussed as part of the case studies in Chapters 6 and 9 and Appendix C.

3.2.3 Summary

Referring to the model from Section 2.3, it can be seen that Design Prototypes are used to represent concepts from various layers of our model: Design Prototypes make use of specific physical hardware for interaction such as paper or TFT screen/mouse (*physical* layer). Paper Prototypes are drawn with stencils in different colours at specific positions of the paper (*alphabetic* layer). These drawings are made up of geometric shapes (*lexical* layer), which represent certain GUI elements, such as buttons

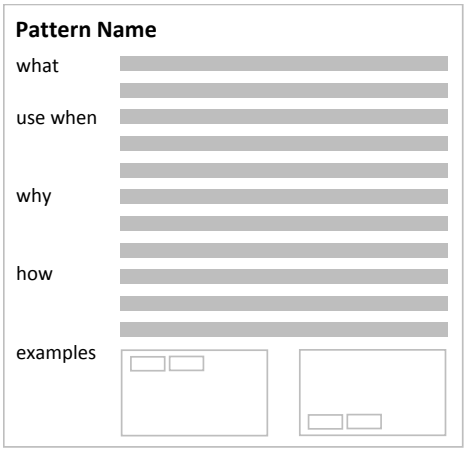

Figure 3.4: UI Design Patterns [85] are represented by prosaic descriptions and examples (schematic depiction).

Layer	Paper Prototype	Electronic Prototype
Application Layer		
Conversation Layer		
Task Layer		
Semantics Layer		
Syntax Layer		
Lexical Layer		
Alphabetic Layer		
Physical Layer		

Table 3.1: Coverage of the layers of the Conversational Virtual Protocol Model by Design Prototypes.

(*syntax* layer). In a wireframe, pushing a button is interpreted as choosing to activate a link (*semantic* layer), which leads the user to another wireframe of the same use case (*conversational* layer). Following the Links in an Electronic Prototype the user finally reaches a "final" wireframe in the Storyboard which coincides with the completion (or cancellation) of the user task (*task* layer).

Table 3.1 provides an overview of the layers that are represented by Paper Prototypes and Electronic Prototypes.

The same observation holds for UI Design Patterns: Table 3.2 shows to which extent the different categories[1] cover the different layers of an IDS. The real complexity even goes further, as individual patterns within these categories span some of the layers of their category, seldom all of them.

This inherent, but implicit complexity of Design Prototypes and UI Design Patterns is one of the reasons for problems surrounding the integration of users into the design of IDSs. Dix [22, p. 301] describes this as a *formality gap* that arises from the necessity for rapid development of *informal* prototypes on the on hand, and efficient *formal*

[1]The names of the categories have been slightly adapted to reflect the terminology used in this work.

Layer	Information Arch.	Navigation	Page Layout	UI Elements	Data Visualization	User Input	Editors	Visual Design
Application Layer						■	■	
Task Layer	■	■	■	■	■			
Conversation Layer	■	■	■	■	■	■	■	
Semantics Layer	■	■	■	■	■	■	■	
Syntax Layer	■	■	■	■	■	■	■	
Lexical Layer	■	■	■	■	■	■	■	■
Alphabetic Layer							■	
Physical Layer							■	

Table 3.2: Coverage of the layers of the Conversational Virtual Protocol Model by UI Design Patterns.

refinements of interaction structures as part of software engineering on the other hand. Chapter 5 will introduce a *structured representation* of Design Prototypes and UI Design Patterns to allow for an integration of the advantages of *informal* UI Prototyping with *formal* software engineering methods.

The next section will discuss the state-of-the-art of UI Prototyping as a basis for understanding the environment in which Design Prototypes are created and UI Design Patterns are applied.

3.3 Requirements Engineering for Interactive Dialogue Systems

In general, RE involves the *elicitation, specification and validation* of requirements [80] (cf. Figure 3.5) and their subsequent *tracing* and *verification* [10]. The elicitation of requirements for IDSs may involve *interviews*, *ethnography*, *use cases* and the creation of Design Prototypes. This work focuses on the role of Design Prototypes in RE. They are used in UI Prototyping for *eliciting requirements* together with users and other stakeholders such as the customer. The prototypes themselves are then used as *specification* artefacts for the elicited requirements, which are typically *user tasks*, *task structures* and the structural and visual/auditory/haptic features of the *UI*. Design Prototypes are used for *validation* as part of User Testing. As we will see in Chapter 4, the informal nature of Design Prototypes complicates the *tracing* of requirements. A formal *verification* of requirements relies on their formal representation, which means that the requirements that are elicited using Design Prototypes need to be translated into a formal representation first.

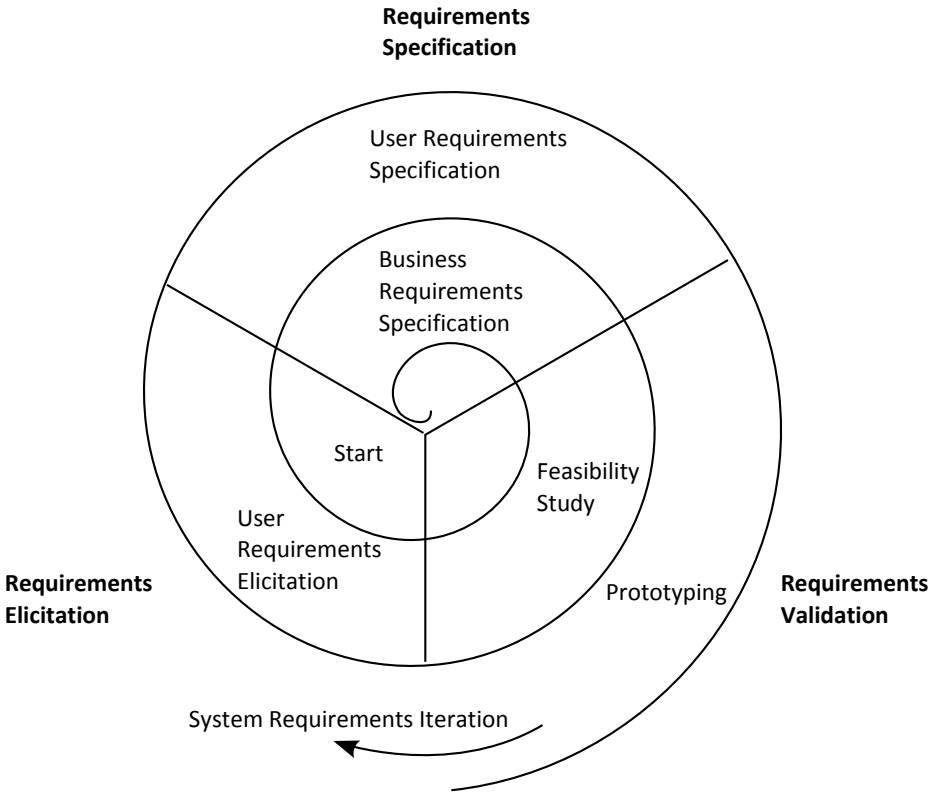

Figure 3.5: Requirements Engineering according to Sommerville 2011 [80] (illustration adapted to the iterations that are of interest for this work).

3.3.1 User Interface Prototyping

UI Prototyping is a prototyping method for creating Design Prototypes and is thus part of the Usability Activities as defined above. It is a common misunderstanding that Usability Activities are mere visual design and testing activities [64]. Instead Usability Activities are important for eliciting user-related and UI-related requirements [64] in early iterations (cf. Figure 3.5) of a spiral software engineering process [7]. As Design Prototypes are used for eliciting requirements, UI Prototyping profits from a combination of a *parallel design* phase [60, 62, 61] and an *iterative design* phase [59] (cf. Figure 3.6).

The purpose of the *parallel design* phase is to explore a set of alternative approaches to satisfying a given use case. The exploration of alternatives allows to choose the best approach out of the given set of Design Prototypes before a merely incremental improvement of the Design Prototype during the *iterative design* phase (see Figure 3.7). Glushko [30] and Buxton [33] describe this as searching for a "global optimum" instead of a "local optimum" in the "design space" or as "getting the right design" and "the design right".

UI Prototyping methods can be categorized according to the process phase in which they are applied, the level of functionality the resulting prototypes provide, the characteristics of the tools used to create prototypes, and the engineering methodology

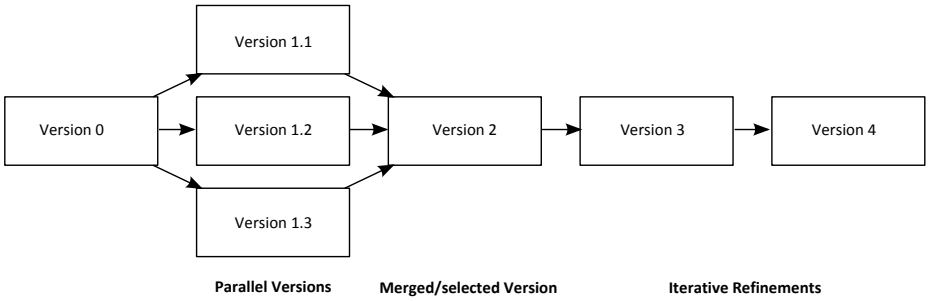

Figure 3.6: A parallel design process allows for exploration of fundamentally different interaction concepts (adapted from Nielsen 1993 [60]).

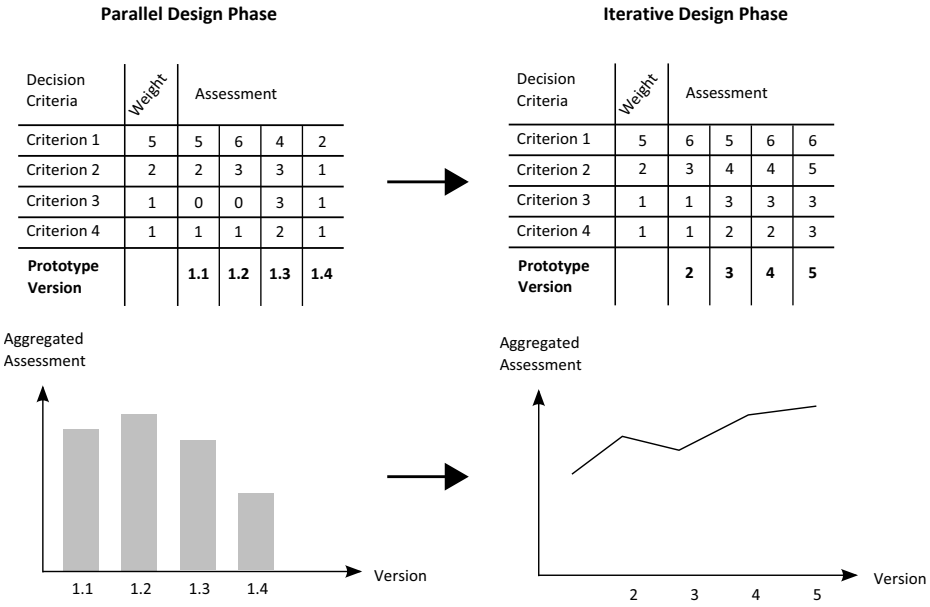

Figure 3.7: Parallel design helps find the right starting point for incremental improvement.

used [73]. The latter suggests a categorization into *paper and pencil, façade tools, interface builders, model-based tools, domain-specific tools* and *actual implementation* [82].

Paper and pencil approaches are *easy to use*, provide *fast turnaround* times for creating prototypes and are best suited for eliciting requirements concerning *task specifications, UI structure and behaviour* [82]. This section introduces specific instances of paper and pencil methods which are used to create the types of Design Prototypes introduced in Section 3.2.

Paper Prototyping

Purpose and timing. Paper Prototyping is usually used in the first iterations of a *parallel design* phase, as Paper Prototypes can be developed with little effort, allowing

for user integration at the same time. It is usually used to elicit *initial* requirements concerning user tasks.

Tools. Paper Prototypes are created by using some kind of stylus to draw on a sheet of paper. In order to make them reusable prototypes (or parts thereof) can be photocopied. If the prototype should be capable of representing minor changes in the state of a UI, smaller pieces of paper can be used to pin the visible results of such changes to the prototype (cf. Figure 3.8).

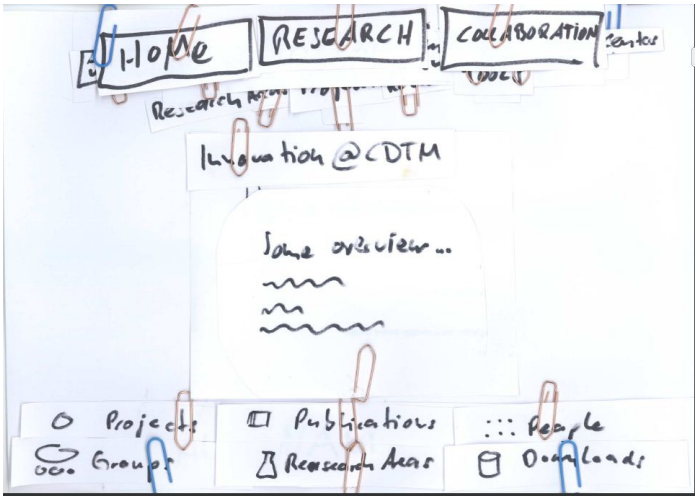

Figure 3.8: Paper Prototype with "changes in state" attached to it.

Roles. Paper Prototypes are usually created by *HCI experts* such as interaction designers, product managers, visual designers or information architects. In addition, different types of *users* can be invited to assist in the creation of prototypes. This is an opportunity to elicit and discuss various requirements for an IDS such as user tasks (which are specific to certain categories of users [65]), UI Design Patterns to be used, visual appearance, etc.

Electronic Prototyping

Purpose and timing. Although being less resource intensive than creating a functional prototype, Electronic Prototypes require significantly more resources to be created than Paper Prototypes. This is caused both by the tools used, but also by additional semantics imposed by Links. Electronic Prototyping is therefore more suitable for late iterations of a *parallel design* phase and early phases of an *iterative design* phase.

Tools. Electronic Prototypes can be created with a variety of different tools, all of which share the property of being able to produce wireframes that can be linked to each other. Although generic tools[2] can be used for creating Electronic Prototypes,

[2]e.g. Microsoft PowerPoint, Microsoft Visio, Adobe Photoshop

more specialized tools exist[3], that provide features such as stencils for GUI elements, a Storyboard view, or the ability to export HTML versions of a prototype. Recent surveys of the growing number of tools can be found in [40] and [46].

Roles. Electronic Prototypes are usually created by *HCI experts*. They can be shared with users and *other stakeholders* such as investors, sponsors or business partners for collecting feedback and provide additional possibilities for *User Integration* as discussed below.

Towards Evolutionary Prototyping

Purpose and timing. Evolutionary prototyping tools integrate Electronic Prototyping into an existing integrated development environment (IDE). They allow for an immediate conversion between informal Design Prototypes and actual implementation. Due to this integration they are well suited for early and late iterations of an *iterative design* phase.

Tools. Prominent examples are Microsoft's Sketch Flow[4], which is integrated with the Microsoft Expression Studio IDE and Adobe's Flash Catalyst[5], which used to be integrated with Adobe's Creative Suite, but has since been discontinued.

Roles. As with Electronic Prototypes, Evolutionary Prototypes are usually created by *HCI experts* and shared with users and other stakeholders. However, Evolutionary Prototyping tools also allow a seamless integration of software engineers and UI designers for the refinement of prototypes.

3.3.2 User Integration

Requirements can be user-dependent (e.g. depending on specific user tasks, context of use or demographic) and user-independent (e.g. typical usability requirements such as the dialogue principles of ISO 9241-110 [21]). Users need to be integrated into the elicitation and validation of the former (cf. UCD). According to Sommerville [80] users and other stakeholders do not know what they really want and express requirements informally. During UI Prototyping adequate *User Toolkits* facilitate contributions by users to the creation of prototypes by being able to experience a possible solution and being able to adjust the prototype to their individual needs. During the evaluation of a resulting Design Prototype, *User Testing* can be used to validate the elicited requirements with users.

[3] e.g. Balsamiq Mockups (http://www.balsamiq.com/), Pidoco (http://www.pidoco.com/), Denim [45] (http://dub.washington.edu/denim/).
[4] http://www.microsoft.com/germany/expression/products/Sketchflow_Overview.aspx
[5] http://www.adobe.com/products/flashcatalyst.html

User Toolkits

Integrating users into the design of products is not a novel notion, neither in RE nor in non-software related product development disciplines. The goal of RE is to identify and formalize requirements. In the case of an IDS some of these are user related (e.g. user tasks, UI Design, usability). User related requirements are *sticky* as users are often unable to communicate their needs in a way that can be used by requirements engineers. In this respect, the term *stickiness of information* describes the "incremental expenditure required to transfer [it] to a specified location in a form usable by a specified information seeker" [38]. User Toolkits are "integrated sets of product design, prototyping, and design-testing tools intended for use by end users. The goal of a toolkit is to enable non-specialist users to design high-quality, producible custom products that exactly meet their needs." [38] They can help with integrating users into the design of IDSs by providing the tools to adjust a given Design Prototype to the specific needs of the user. Klemmer et al. [41] describe Papier-Mâché, a User Toolkit for building multimodal IDSs, which allows users to customize the behaviour of the IDS by associating the detection of an input event on a given input channel with the generation of an output event for a given output channel.

As part of engineering processes, User Toolkits can be used to transfer *need-intensive* product development tasks to the user, who owns sticky information with respect to user related requirements. Once the requirements are fixed, *solution-intensive* tasks can then be conducted within the regular software engineering process. [38]

The UI Prototyping tools mentioned in Section 3.3.1 are, in fact, User Toolkits, because they provide users with tools that allow them to engage in design and innovation activities. Design Prototypes facilitate a conversation with the user's mind [14] to extract sticky information for the elicitation of requirements. The *motivation* of a user to take part in the process is crucial. Besides *external factors* such as job assignment, reputation or customer needs, there are *internal factors* such as identification, enjoyment, or intrinsic motivation that need to be taken into account for the design of a User Toolkit and its application [72]. In case of User Toolkits that are used to customize existing products or services (e.g. mobile applications [24], energy efficient housing solutions [67]) according to user needs, the motivation of participating users can often be attributed to *internal* factors. Motivating users to take part in UI Prototyping more relies on *external* factors (e.g. payment) unless it is possible to activate *internal* factors. Leimeister et al. [44] describe different ways of activating external motivation factors in the context of idea competitions. A recent study furthermore suggests that the overall quality of ideas can be improved by collaboration among the participants of the idea competition [5]. User integration with toolkits for idea competitions are not widely used in practice, yet [6].

User Testing[6]

Due to several issues with the integration of users into the design of IDSs, users are mostly integrated for *validation* activities, i.e. user testing, observation, questionnaires, interviews, focus groups, logging actual use and user feedback [59]. User Testing can be defined as a set of "techniques in which users interact systematically with a product or system under controlled conditions, to perform a goal-oriented task in an applied scenario, and some behavioural data are collected" [89]. User Testing is widely accepted and applied in software engineering projects in a variety of industries [89]. As Design Prototypes are artefacts for eliciting (and representing) UI requirements, the purpose of *User Testing* for Design Prototypes is to *validate* whether a given set of requirements reflects *actual* requirements.

User Testing for GUI-based Design Prototypes can be conducted for both types of Design Prototypes introduced above. User Testing is conducted based on *Test Plans* [59]. Test Plans describe the *goals*, a set of *test cases* (the user tasks to be tested), the *start* and *stop* condition of each test case and several other logistic properties. There are different approaches on how to conduct the test itself.

In case of User Testing for Paper Prototypes, the tests are usually conducted in the presence of an experimenter, who is responsible for introducing the tester, helping the tester during the test and debriefing the tester in the end. It is not uncommon to conduct the test in a lab environment and to record the test on video. The videos are then later reviewed and the key results documented in written form. An alternative approach is the so called "Discount Usability Testing" method [43] that does not rely on a lab environment. Instead of recording the test, the experimenter or a third person writes down the key results during the test. A method that can be used to extend the usefulness of a test is to ask the user to *think aloud* while conducting the test.

Electronic Prototypes can either be tested with the same techniques as Paper Prototypes, or they are tested using remote usability testing. In the latter case testers log in to a web-based user testing platform that allows the tester to conduct the test without the help of an experimenter. The advantage is the increased scalability in the number of potential test users, the increased flexibility in terms of location, time and duration of the test, and the increased validity due to the tests being performed in the tester's environment (instead of a lab). The disadvantages are the reduced intensity of direct observation, the reduced possibility for guiding the tester during the test, and a limitation of potential testers to individuals who are proficient in using the internet.

[6] User Testing is also referred to as *Usability Testing*. In this work the term User Testing is preferred, as the method does not only focus on *Usability*, but more generally on *User*-related requirements, e.g. user tasks, user experience, usability, etc.

3.3.3 Summary

UI Prototyping belongs to Usability Activities and is applied in RE for eliciting user-related and UI-related requirements. Integrating UI Prototyping into a parallel design process, allows for simultaneous exploration of design alternatives in order to design a better product. UI Prototyping methods for GUI-based Design Prototypes are supported by readily available tools. Some of these tools also provide the functionality of User Toolkits for the elicitation of user-related requirements for IDSs. Elicited requirements are validated by engaging users in Usability Tests.

It was rather difficult to explain how to use and interact with the prototype.

study participant

4 Study: Challenges for Integrating Users

Chapters 2 and 3 introduced a conceptual model of HCI for IDSs and methods and artefacts that are used to design IDSs. This chapter summarizes the design and results of an exploratory case study that has been conducted in 2009 with the goal of examining the advantages and disadvantages of *Paper Prototyping* and *Electronic Prototyping* under the assumption of integrating non-experts into the process. The results were used for generating hypotheses with respect to the challenges for integrating users into the design of IDSs. The following study report is structured according to the case study research process outlined in Runeson and Höst 2009 [70].

4.1 Introduction

A prototypical User Toolkit for Electronic Prototyping and User Testing was developed at the Center for Digital Technology and Management (CDTM) in 2008 [2, 34]. It was subsequently applied in an interdisciplinary graduate course on *Advanced Topics in UI Design* at CDTM in which 21 students took part in 2009. As part of the course students developed Design Prototypes of GUI-based IDSs both with traditional Paper Prototyping techniques as well as Electronic Prototyping based on the toolkit and a professional Electronic Prototyping tool by Pidoco GmbH.

Problem Statement

Paper Prototyping is a well established prototyping method. Electronic Prototyping is supported by a growing number of academic and professional tools [40]. Both methods allow for the integration of users. Still, users are usually integrated into software engineering projects only occasionally. One of the reasons often mentioned is the high costs caused by activities for user integration. It is unclear, however, if the assumption of high effort for integrating users is also valid for user integration based on Electronic Prototyping.

Research Objective

The objective of the case study was to identify advantages and limitations of Electronic Prototyping and Paper Prototyping with respect to the integration of non-specialists in the design of UIs of IDSs. In this respect the expected result of the

study is a set of findings that motivate *hypotheses* for further research towards a better integration of users into the design of IDSs.

Study Partners

CDTM[1]. The Center for Digital Technology and Management is a joint institution of Ludwig-Maximilians Universität München and Technische Universität München. It offers a graduate study program in Technology Management for master-level students in computer science, electrical engineering, and business administration. Besides these fields of study, students from other fields of study are accepted as well if they proof sufficient knowledge in one of these or related fields of study. 20 students per term are admitted to the program. Teaching at CDTM is focused on *interdisciplinarity, team work* and the *integration of theory and practice*. At the time of the study, CDTM was attended by approx. 80 students and run by a management team of 2 scientific directors and 10 doctoral candidates.

Pidoco GmbH[2]. Pidoco is a Berlin-based start-up company, which was founded by alumni of CDTM and Hasso-Plattner-Institut[3] in 2008 and which received funding from High-Tech Gründerfonds. Pidoco offers a subscription-based web service for Electronic Prototyping and User Testing in a Collaborative UI Design approach. At the time of the survey, Pidoco was run by a team of four founders. Pidoco supported the case study with free accounts for the UI Design projects.

4.2 Case Study Design

Research Questions

This study examines

> **RQ 1** What are advantages and limitations of using Design Prototypes for eliciting and validating requirements?

In order to learn about the advantages and disadvantages Design Prototypes for eliciting and validating requirements, the case study focuses on the *suitability* of Design Prototypes (more specifically paper prototypes and electronic prototypes) for UI Prototyping (cf. Section 3.3.1). The *suitability* is measured along two dimensions: The *expressiveness* of the artefacts and their *descriptiveness*. Expressiveness is important during the parallel design phase in order to explore alternative approaches to satisfying a given use case during *parallel* design phases. Descriptiveness is important for the traceability of requirements during *iterative* design phases.

[1] http://www.cdtm.de

[2] Until September 2009 the company and its product was named RapidRabbit. The service can be found at http://www.pidoco.com/

[3] http://www.hpi.uni-potsdam.de/

RQ 1.1[4] *Are Design Prototypes sufficiently descriptive for being used as requirements artefacts?*

With respect to the study environment, this question points at the quality of the prototyping artefacts in terms of how easy it is to understand them and to hand them over to previously uninvolved collaborators. This gives some hints to possible advantages or disadvantages with respect to iterative design as part of the requirements engineering phase.

RQ 1.2 *How does the prototyping method used affect the expressiveness of Design Prototypes?*

Electronic tools may limit the expressiveness of artefacts designed with these tools both by the types of operations that can be conducted with the tools and the effort needed to learn how to master the tools. Artefacts that are created for User Testing with the help of electronic tools may limit users to expected behaviour.

Case and Subjects Selection

The *case* under investigation is the practical training that accompanied the graduate course *Advanced Topics in UI Design.* The practical training took place in four different project teams, which are the *units of analysis.* The *subjects* taking part in the study have been selected by availability and prior experience: Students applied to the course and the assignment of students to project teams focused on creating project teams with multi-disciplinary backgrounds in order to allow for cross-discipline interactions within the teams. Each project team was assigned to one of two possible UI design projects that were informally specified as a set of use cases. The goal of each project was to come up with a Design Prototype by going through two prototyping and validation cycles. After the first prototyping and validation cycle, teams would hand over their Design Prototypes, test plans and other artefacts to another team to intensify the impact of the suitability of Design Prototypes for requirements elicitation and validation.

Data Collection Procedures

The study is based on a triangulation of first-degree and second-degree methods [70, p.144]: Subjects were invited to three rounds of fully-structured interviews with both open and closed questions. The interviews were conducted online using a simple form-based questionnaire (see Appendix B). The interviews took place before the first design phase ("pre-course interview"), after the first evaluation phase ("mid-term interview") and after the second evaluation phase ("final interview"). The project phases are discussed in detail in Section 4.3.1. Besides the interviews, design and evaluation artefacts (Design Prototypes, test plans, test reports) were collected from the subjects at the end of each design and evaluation phase. Independent analysis

[4]The numbering of the research questions reflects their relation to the top level research questions described in Chapter 1. RQs 1.1-1.2 are refinemens of RQ 1, RQs 2.1-2.3 of RQ 2 and RQs 3.1-3.3 of RQ 3.

of these work artefacts by the experimenter was used as a second data source for answering the research questions.

Step 1: Before the projects commenced, students were asked for their field of study, their preferences towards the available projects, their prior experience with relevant topics (UI Design, visual design, User Testing, Collaborative UI Design), and their prior experience with HTML and graphics tools. Students were also invited to leave text comments.

Step 2: After the first design iteration, the students were interviewed for their preliminary experience. *Closed* questions concerned

- the number of people per team who were involved in the actual design of Design Prototypes,
- a scaled rating of how easy it was to create the Design Prototypes collaboratively,
- the estimated average amount of time necessary to create an individual mock-up,
- the number of test users for User Testing and
- the estimated average amount of time necessary to conduct a user test for a single user task.

Open questions concerned the interviewees' perceptions with respect to advantages and disadvantages concerning the UI Prototyping methods and the use of Design Prototypes for User Testing. Again, additional text comments were collected. In addition to the interview, the project teams were asked to hand in their Design Prototypes, test plans and test reports.

Step 3: After the second and final design iteration, the students were interviewed for their experience since the last interview and overall perceptions. Besides the questions from the last interview round, which were repeated, an additional *closed* question asked for a scaled rating of how easy it was to understand the design and test artefacts that were handed over from the last iteration. Optional text comments and the design and test artefacts were collected again. Finally, all project teams delivered a final report summarizing their results and conclusions.

Analysis Procedures

As the case study approach is a flexible research approach, data analysis was carried out in parallel with the data collection in order to allow for updating questionnaires for subsequent interview rounds and thus being able to further investigate preliminary findings. The analysis was based on grouping interview answers in order to draw conclusions and to generate hypotheses. The analysis was conducted in two steps:

- Step 1 addresses the research questions by subject interviews
- Step 2 addresses the research questions by content analysis

Step 1: After the interviews were conducted, the *closed* answers were analysed by grouping the answers by UI Prototyping method and calculating average responses.

The answers to open questions were grouped by UI Prototyping method and positive or negative tendency.

Step 2: The design and test artefacts as well as the final reports that were handed in were examined for additional indicators concerning the research questions. As the artefacts were not homogeneous, the analysis was unstructured and focused on identifying evidence for the responses in step 1.

Validity Procedures

Validity is difficult to achieve in this qualitative case study. Threats to validity may be caused by *constructs* (students from different fields of studies will have a different understanding of the subject matter), *internal factors* (the study is heavily influenced by the tools being used), *external factors* (findings from a course setting are hardly generalizable to industrial settings) and *reliability* (the project setup and the project switching scheme is not easy to explain and thus potentially hard to repeat by other researchers). Nevertheless, there were some procedures in place for improving validity:

- *Triangulation* was introduced by supporting the interviews with artefacts-based content analysis.
- Thanks to a flexible study design the project teams could be assigned after the first interview. This allowed for the definition of project teams that share competencies and roles similar to professional project teams.
- All subjects were exposed to both project topics and to both UI Prototyping methods by introducing scheduled switches between projects (see Section 4.3.1).

As the scheduled switching between projects forced the teams to hand over their work artefacts, the impact of the expressiveness and descriptiveness of the artefacts on the work in subsequent iterations could be intensified. This was necessary in order to identify effects which would otherwise be hardly perceivable due to the internal factor of a limited number of iterations.

4.3 Results

As a result of the interviews, the *limited seamlessness* of using Paper Prototypes in the design process and the *increased flexibility* in terms of iterative collaboration with Electronic Prototypes was identified.

4.3.1 Case and Subject Description

The units of analysis are four UI Design projects, each two of which are based on the same project objective: projects 1 and 2 targeted the design of a website for an information task. Projects 3 and 4 targeted the design of a shopping widget for an existing website. Both projects were defined by the course administration with the intention of providing sufficient options for designing both the interactions as well

as the GUI. The projects also differed in their underlying methodologies: Projects 1 and 2 made use of Paper Prototyping and classical user testing, while projects 3 and 4 made use of Electronic Prototyping and User Toolkit-based testing. In order to investigate the research questions with a certain level of validity (see above) it was decided to let the subjects experience *both* projects and *both* methodologies by imposing a scheduled switch of project teams between projects and methodologies. Teams had to switch projects and methodologies during the course of the term along two design and two evaluation phases (Figure 4.1 provides an overview of the scheduled switches). The switches also helped in increasing the impact of the level of documentation of the work artefacts (Design Prototypes, test plans and test reports) as these artefacts needed to be handed over from team to team.

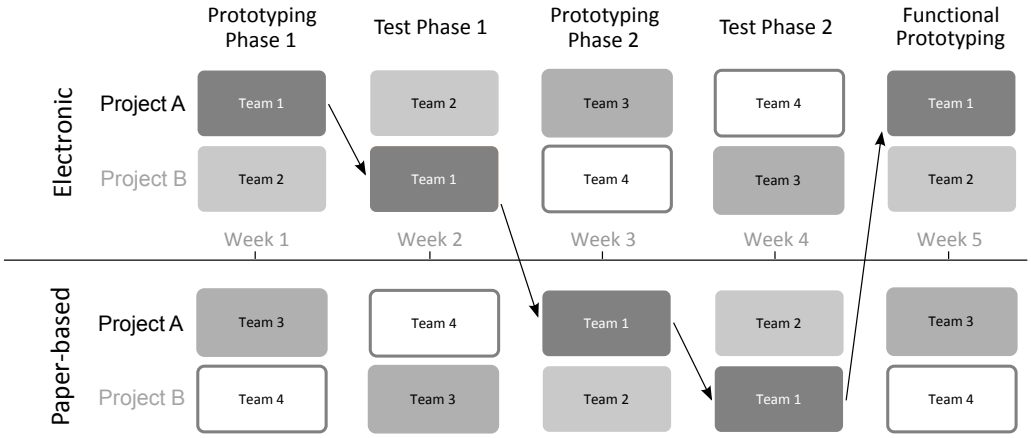

Figure 4.1: The teams were assigned to different projects and different prototyping methods which forced the teams to hand over prototypes and validation results.

In total 21 students took part from the fields of computer science (5), media computer science (3), mathematics (3), business administration (3) and various other subjects such as economics, industrial design and political science (7). The results of the first interview round among the subjects, which asked the subjects to provide a self-assessment, reflects the heterogeneous background of the subjects[5]:

- Prior experience with UI Design: 7 with practical experience, 12 without
- Prior experience with graphical design (in general terms): 3 with practical experience, 16 without
- Prior experience with User Testing: 1 with practical experience, 18 without

Based on these results the subjects were assigned to the projects (cf. Table 4.1).

[5] 1 student left the course and 3 students joined the course after this survey.

	No.	1	2	3	4	5	6	7	8	9	1	11	12	13	14	15	16	17	18	19
Subject	Field of Study	C	E	B	E	O	C	C	O	O	O	O	C	C	E	C	O	O	C	C
	Team	1					2					3					4			
	Project	Information Site					Shop Widget					Information Site					Shop Widget			
Experience	UI Design	◐	◑	◑	◐	◐	◐	◑	◑	◑	◑		◑	○	◑	◑	◐	◑	◐	◐
	Visual design	◐	◑		○	○		○	○	◑	◑		○	○	○	◑	◐	◑	◐	○
	User Testing	◑	○	○	◑	○	◐	◑	○	◑	○			◑		◑	○	○	○	○
	Collab. UI Design				○	◑												○		
	HTML	◐	◐		◐	○	●	◐	◑	◑	○	◑	○	●	◑	○		◑	●	●
	Visual design tools	○	○		◐	◐	●	◑	○		◑	○		○	◑	◑	●	◑	◐	◐

Table 4.1: The teams were set up to reflect collaboration in multi-disciplinary teams (B: Business Administration, C: Computer Science including Media Computer Science, E: Electrical Engineering, O: Other) with different levels of prior experience (<empty>: none, ◑ : knows what the term relates to, ○: basic knowledge, ◐: practical experience, ●: expert).

4.3.2 Are Design Prototypes sufficiently descriptive for being used as requirements artefacts? (RQ 1.1)

Design Prototypes are not self-explanatory. Students were asked how easy it was to understand the Design Prototypes handed over by other teams during the 2nd and 3rd interview round (cf. Figure 4.2). 6 students working on paper prototypes answered with "It was hardly possible", while none of the students working with electronic prototypes provided this answer. This can be explained with the fact that Electronic Prototypes provide additional semantics by integrating a simple model for the *semantic* layer (cf. Chapter 3). Nevertheless, 31 out of 40 responses were critical ("hardly possible", "rather hard") towards the descriptiveness of Design Prototypes.

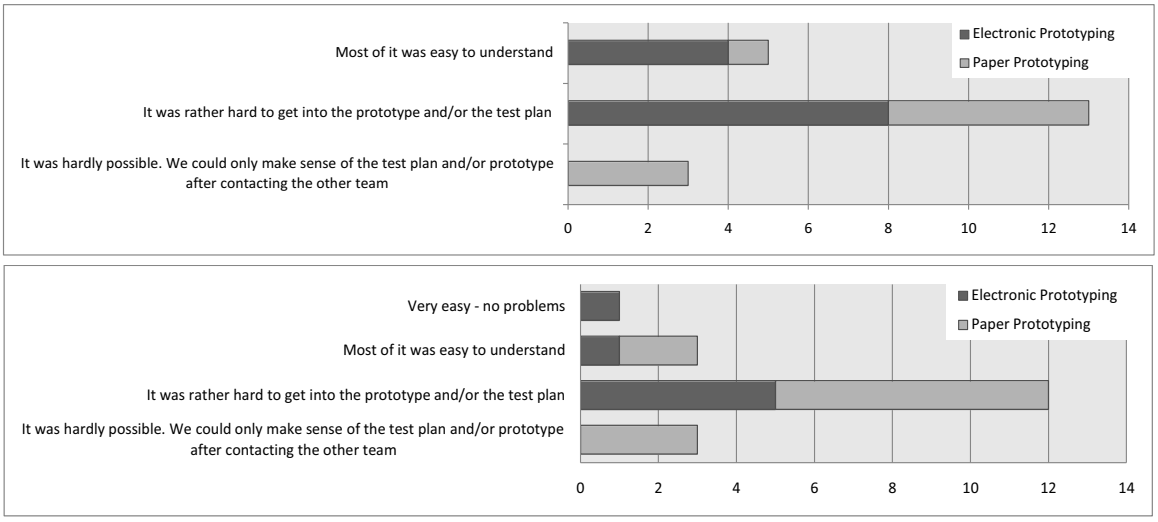

Figure 4.2: Design Prototypes are not self explanatory - responses to question "How easy was it to understand the *other team's* test plan and prototypes?", interview round 1 (top) and round 2 (bottom).

This is also reflected by comments by individual participants on Paper Prototyping and Electronic Prototyping:

Paper Prototyping:

> "Very confusing, we never were sure which buttons could be used and where they would link to."
>
> "Not very intuitive, often hard to understand where to click/navigate."
>
> "It was rather difficult to explain how to use and interact with the prototype."
>
> "Working into the prototypes of other teams was very hard and partly impossible. You basically need someone to explain what each page denotes and how they link to each other. Without this knowledge, you cannot conduct a meaningful user test, because you do not know, what is supposed to happen once a user clicks somewhere." [translated from German]

Electronic Prototyping:

> "The electronic prototypes already had some basic functionality, making them much easier to understand. Meetings in-person were not necessary to work on the prototypes."
>
> "The Electronic Prototyping was in my point of view easier to use and easier to imagine what the other team might have had in mind while designing their prototype. We did not have to meet in person, but everyone could go through it by himself and than contact the team and share his/her experience."
>
> "Fast and easier to modify than paper, for testers it was also easier to understand and more intuitive than paper."

Design Prototypes allow for collaboration. Students were also asked how easy it was to collaborate on the design artefacts and to rate it on a scale from 1 (very easy) to 5 (very hard). 20 out of 40 responses were positive ("very easy", "easy") and another 18 responses were neutral (see Figure 4.3). Only 2 responses indicated that it was hard to collaboratively work on the design artefacts.

4.3.3 How does the prototyping method used affect the expressiveness of Design Prototypes? (RQ 1.2)

Paper Prototypes provide a less restricted visual design space. Another question the interviews examined was the *expressiveness* of the artefacts generated with either prototyping method. Students were asked to provide unstructured comments concerning what they specifically liked and disliked about either method. The results indicate that paper prototypes provide more extensive expressiveness as compared to electronic prototypes. This is probably due to the fact that the toolkit

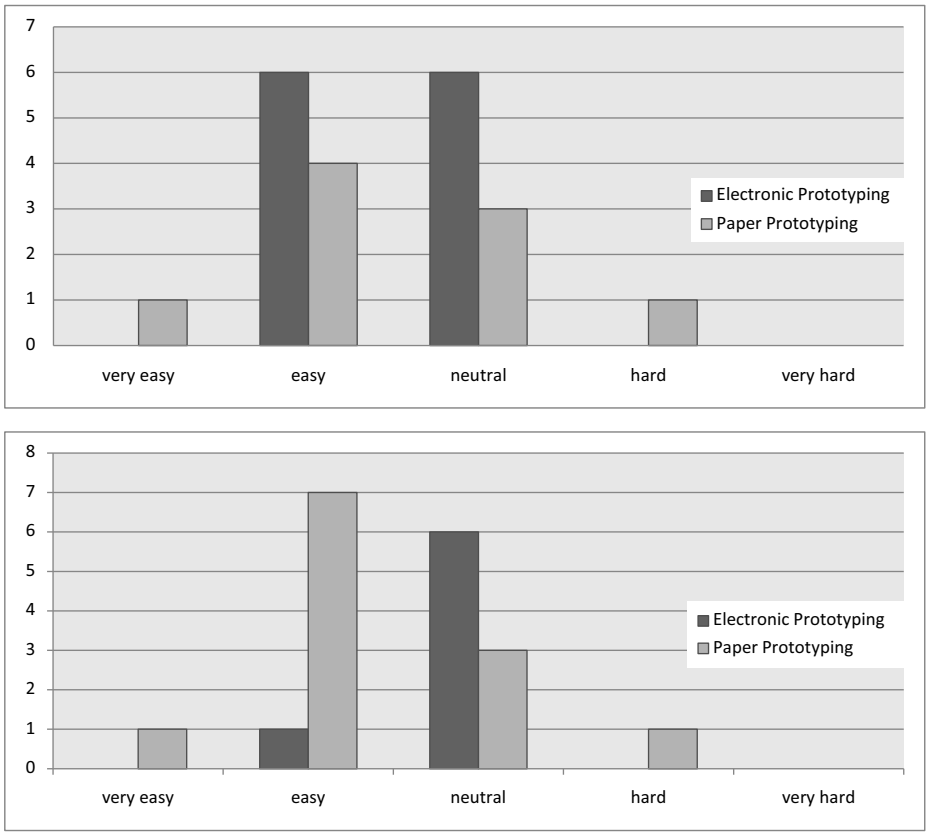

Figure 4.3: UI Prototyping is easy to apply and allows multi-disciplinary teams to collaborate - responses to question "How easy was it to work together on the *design*?", interview round 1 (top) and round 2 (bottom).

for Paper Prototyping (paper and pencil) does hardly restrict the creativity of the creator.

This is reflected by individual comments by participants of the study concerning the use of Paper Prototypes:

> "Possibility to draw every kind of geometric shapes (often not possible with [the Electronic Prototyping toolkit]), easy to change/add design features"
>
> "Paper Prototyping allows to create very basic prototypes more quickly than Electronic Prototyping because there are almost no constraints."
>
> "Freedom of creativity"
>
> "More flexibility, more creativity (not restricted to the set of provided features, conditional output possible,...)"

Electronic Prototypes are more realistic. Due to its increased fidelity, Electronic Prototypes allow for more realistic Design Prototypes. Test users can interact with the prototypes autonomously and get an impression of the desired interaction experience. Again, some statements into that direction:

"A real environment (mouse, screen, keyboard) could be used to do the testing."

"It actually "works", so if you click on a button, something happens!"

"The user could already use the webpage as if it was real."

"I think it's easier for the user to understand what the final GUI might look like."

4.3.4 Evaluation of Validity

As the study is used for hypothesis generation, validity is not as critical as in the following studies. However, some pre-cautions have been put in place as described above. The course environment and the artificial projects are threats to the *construct validity*. The effects identified in these projects can hardly be generalized to a realistic setting. As the students involved in the projects belong to the top ~10% of their individual field of study and the inidividual teams were staffed with students from different fields of study, who were capable of filling in the roles of software engineers, HCI experts, etc., a certain resemblance in terms of proficiency and interdisciplinarity can be assumed. In addition, the tools used for prototyping are used in a similiar fashion in professional projects and the students were coached for their use by academic and industry experts. There are also a number of threats to *internal validity*: The tools that have been used, the project topics, and the scheduled switches between projects are all influencing factors. Triangulation of data sources was used to resolve these threats to some degree. The *external validity* is given as UI Prototyping is part of numerous professional software development projects. It can be assumed that the effects described by this study (with the limitations incurred by the other validity aspects) can be found in professional projects, too. *Reliability* depends on whether the study environment can be replicated using similar tools and a similar interdisciplinary class environment.

4.4 Conclusions

Summary of Conclusions

The results indicate:

1. Paper Prototypes support creativity better than Electronic Prototypes. Electronic Prototypes are generally perceived as being less expressive than Paper Prototypes.
2. It is easy to collaborate on the design of Paper Prototypes and Electronic Prototypes and to validate the artefacts with users.
3. Paper Prototypes and Electronic Prototypes are not self explanatory and are therefore not well suited as requirements *specification* artefacts.

These effects are mainly caused by the nature of the artefacts generated by each method and the tools used. Paper prototypes do not explicitly contain any kind of representation of a conversational model. Instead, the relationship between individual screens needs either be documented manually by the creator(s) or the creator(s) need to be present every time the prototype is used.

On the other hand electronic prototypes are created with tools that make it harder for creators to be as free in their design decisions as with paper, pen and scissors/tapes. This effect could probably be alleviated by introducing collaborative, digital interaction techniques. What is important is that a primitive design toolkit, as it is used for Paper Prototyping, is necessary for allowing for a maximum of expressiveness during early iterations.

Relation to Existing Evidence

An earlier empirical study from 2002, that was conducted among practitioners from different software engineering domains suggested that requirements are mostly modelled using informal notations. [54]. In current practice users are integrated at specific project milestones at most [88, 87].

This study investigated aspects of a specific type of informal requirements artefacts: Design Prototypes. It suggests that the integration of users into the design of IDSs might be in part hindered by the limited descriptiveness of the informal Design Prototypes themselves.

Impact/Implications

The findings from this study suggest that Design Prototypes are not self explanatory. The reason may be found in the theoretical discussion from Chapters 2 and 3: It indicates that IDSs can be modelled using a multi-layered virtual protocol model. Design Prototypes are used to elicit requirements for IDSs. Therefore, the findings of this study and the previous theoretical discussion lead to the following hypothesis that will be discussed in the remainder of this work:

> *Design Prototypes can be understood as an unstructured representation of numerous requirements concerning an IDS. Although the Design Prototypes seem to represent only one aspect of an IDS– the visual elements of a UI–, they implicitly model requirements across all layers of the model defined in Section 2.3.*

Limitations

This study was conducted in an academic course environment, therefore lacking many of the specifics of professional software engineering projects. The projects themselves were very small in scale. As Design Prototypes are usually generated in very early phases of the requirements engineering process, professional projects do not defer to much in terms of the size of the prototypes[6]. A clear limitation of the study is the

[6]See Chapters 4 and 9 for studies in a professional setting.

scale of the study itself. The study was conducted with only 21 participants, therefore it does not possess any statistical significance. It would be to weak to deduce final conclusions. It is intended, however, as being sufficient for generating hypotheses that are taken into closer consideration in the next chapters.

> Experts of different disciplines have to cooperate. For their communication and for the documentation of the development they need system descriptions that are sufficiently precise and detailed.
>
> *Manfred Broy et al.*

5 Structured Representation of Design Prototypes and UI Design Patterns

The study presented in Chapter 4 suggests that Design Prototypes support collaborative design in interdisciplinary teams and the integration of test users. However, it also provides evidence that the informal nature of Design Prototypes and their use for requirements elicitation in IDS projects causes issues that are generally known from RE for software-intensive systems [13, p. 3]:

- *"Requirements are not completely and accurately identified and understood by the application expert."* The study suggests that Design Prototypes are hard to interpret and thus the requirements contained in them are hard to trace in an iterative design process.
- *"Requirements are not correctly specified, although completely and accurately identified and understood."* The informal notation of Design Prototypes does not allow for an analysis with respect to their correctness. Correctness per se is not defined for these artefacts.
- *"Requirements are correctly specified, using informal techniques that are not properly interpreted and conceived by the system designer or the implementer."* As the study suggests, the requirements reflected by Design Prototypes are implicitly contained in the Design Prototypes themselves. They need to be extracted from the Design Prototypes which is a manual task with unclear procedures. A tracing of the requirements is hardly possible.

Working towards the goal of integrating users more seamlessly and continuously into the design of IDSs, a first step is to provide the theoretical basis for a better integration of the design artefacts and their requirements. A structured representation of Design Prototypes and their requirements is necessary to allow for the application of more formal methods of software engineering. More specifically, a discrete mathematical notation will be used to define the structural and behavioural aspects of Design Prototypes. This will allow us to map otherwise ambiguous elements of prescriptive theories such as UI Design Patterns to a structured representation in order to better document and trace requirements. This chapter introduces such a structured representation. It will subsequently be used as a theoretical basis for the workflow and tool-support in Chapter 7 and the case studies in Chapters 6 and 9.

5.1 Notation and Concepts

The following sections make use of the description and development techniques for ISs that are offered by FOCUS [13]. It allows us to model the layers of the conversational virtual protocol model and their exchange of messages using *interface specifications* and *streams*. We will use *auxiliary functions* to model the structural dependencies between different layers of Design Prototypes. This section lists the concepts that are used in the remainder of this work. More detailed information can be found in [13].

5.1.1 Mathematical Notation

Besides standard mathematical notation, the following definitions apply:

$\mathbb{N}$ denotes the set of *natural numbers* with $\mathbb{N} = \{1, 2, \cdots, n, n+1, \cdots\}$.

$\mathbb{N}_0$ denotes the set of *natural numbers* including $\{0\}$, i.e. $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

$\wp(A)$ denotes the *power-set* of a given set A, i.e. $\wp(A) = \{B \mid B \subseteq A\}$.

$\exists_1$ denotes the *unique existential quantification*, i.e. $\exists_1 x P(x) \implies (\exists x P(x) \wedge \forall y(P(y) \implies x = y))$.

This work uses the shorthand notations for domains and ranges of functions as defined in [13]:

$f \in A \mapsto B \implies \text{dom}.f = A \wedge \text{rng}.f = \{f(t) \mid t \in A\}$

5.1.2 Streams

The formalisms in this work are based on a small subset of FOCUS. The formal definition of a stream, its basic operations and its motivation can be found in [13]. The following notation is used:

Let M be a *set of messages*. M^ω, M^∞ and M^* denote the set of all *streams*, all *infinite streams* and all *finite streams* over the set M.

$\langle m_1, m_2, m_3, \cdots, m_n \rangle$ denotes a *stream* of n messages.

$\langle\rangle$ denotes the *empty stream*.

$\#s$ denotes the *length* of stream s.

$s.n$ denotes the nth message of stream s.

$\frown \in M^\omega \times M^\omega \mapsto M^\omega$ denotes the *concatenation operator*. It is defined by:

$$(s \frown r).k = \begin{cases} s.k & \text{if } 1 \leq k \leq \#s \\ r.(k - \#s) & \text{if } \#s < k \leq \#s + \#r \end{cases}$$

5.1.3 Interface Specification

Design Prototypes are complex with respect to their structure, but simple with respect to their behaviour. This work uses *untimed, infinite streams* for representing the exchange of messages between user and IDS, i.e. it does not take into account any kind of timing effects. Section 5.3.1 introduces a data model for representing the structure of Design Prototypes. Design Prototypes model the interaction between user and system with a simple set of state transitions (see below). The formalization of interface specifications will therefore be based on *equational specifications* that allow for the representation of interactions as *state transition rules*.

A key element of equational specifications is the *stream processing function* f that describes the mapping between input (I) and output (O) messages:

$$f \in I^{\omega} \mapsto O^{\omega}$$

Parts of the specification that follows below represents *I/O transitions* that work on *I/O states* only, i.e. no encapsulated states are needed for calculating outputs for arbitrary inputs. In order to represent the conversational interaction that is modelled by a Design Prototype, this work will also make use of *encapsulated states*, more specifically *local states*. Local states can be helpful for defining state-aware stream processing functions by using the local state ($currentState \in \Sigma$) as an additional function parameter:

$$f \in \Sigma \times I^{\omega} \mapsto \Sigma \times O^{\omega}$$

5.2 Conceptual Model of IDSs

This section introduces a conceptual model for IDSs as a basis for the definition of a reference architecture for GUI-based Design Prototypes. It is based on the virtual protocol model introduced in Chapter 2. The conceptual model will be the basis for the definition of a reference architecture for GUI-based Design Prototypes in Section 5.3.

Overview of the conceptual model

In a black-box view the conceptual model can be divided into two *components*: The human user and the IDS (Figure 5.1). Let S be the set of n components in the glassbox view of the IDS with $S = \{S_i \mid i \in \{1; \ldots; n\}\}$ and I_{S_i} and O_{S_i} the set of external input and output communication channels of S_i. Then the black-box view of the reference architecture contains $I = \bigcup_{i=1..n} I_{S_i}$ and $O = \bigcup_{i=1..n} O_{S_i}$ input and output communication *channels*.

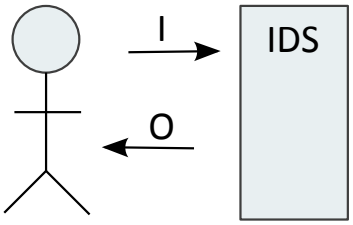

Figure 5.1: Black-box view of the conceptual model.

Layers

The conceptual model describes a layered architecture. Each layer L_n exports an interface I_{n-1} to layer L_{n-1} and an interface O_n to layer L_{n+1}.

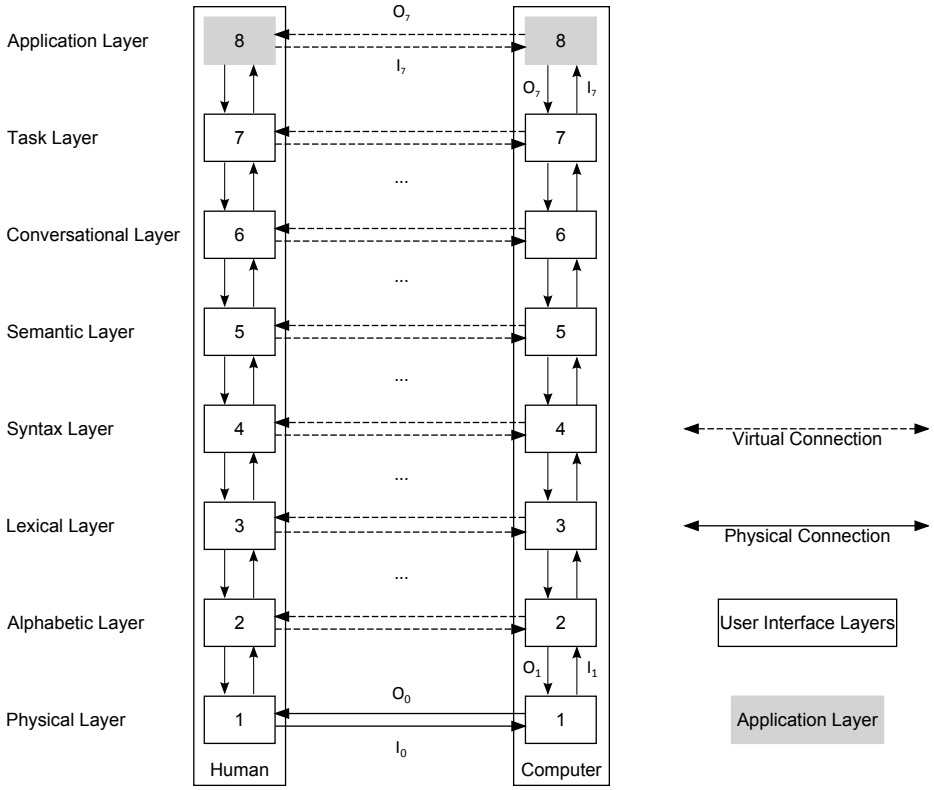

Figure 5.2: Layers and interfaces of the conceptual model.

The layers are those described in the virtual protocol model for IDSs in Chapter 2, namely the *Physical Layer* L_1, the *Lexical Layer* L_2, the *Syntax Layer* L_3, the *Semantics Layer* L_4, the *Conversational Layer* L_5, the *Task Layer* L_6, and the *Application Layer* L_7 (Figure 5.2).

Interfaces

Since the conceptual model is based on a virtual protocol model, each layer virtually exports a channel to its respective counter part on the other stack, i.e. each layer L_n on the system stack exports its interface I_{n-1} to layer L_n on the user stack and

each layer L_n on the user stack exports its interface O_{n-1} to layer L_n on the system stack. The export is called *virtual*, as there is no physical connection between layer L_n on the user stack and layer L_n on the system stack for $n > 1$. Instead all messages that are exchanged between layers are actually relayed through the lower layers and physically transferred through communication layer L_1.

Due to the layered architecture, in which direct communication is exclusively handled by the communication layer L_1, I and O of the composite interface specification for the blackbox view of the IDS are the sets of external input and output channels of the subcomponents of L_1.

The interface of a given layer L_n with input channels I_{n-1}, O_n, output channels I_n, O_{n-1} and message types $T_{n_1}, T_{n_2}, T_{n_3}, T_{n_4}$ can be described as:

L_n

in	$I_{n-1} : T_{n_1}, O_n : T_{n_2}$
out	$I_n : T_{n_3}, O_{n-1} : T_{n_4}$
$f(I_{n-1}, O_n) = (O_{n-1}, I_n)$	
where f so that $\forall i \in T_{n_1}, o \in T_{n_2}$:	
$f(\langle i \rangle, \langle o \rangle) = (f_{L_{n_1}}(i), f_{L_{n_2}}(o))$	

The auxiliary functions $f_{L_{n_1}}$ and $f_{L_{n_2}}$ describe the behaviour of the layer on valid inputs by mapping from the input type to the output type. This generic interface specification will be re-defined for each layer below to include its specific behaviour.

5.3 A Reference Architecture for GUI-based Design Prototypes

This section introduces a reference architecture for Design Prototypes that follows the previously introduced conceptual model for IDSs. It will allow us to have a more structured understanding of Design Prototypes and to map otherwise ambiguous elements of prescriptive theories such as UI Design Patterns to a structured representation in order to better document and trace requirements.

The Design Prototypes that are in the focus of this work represent IDSs across all device-independent UI layers, i.e. from Lexical Layer to Task Layer. However, the techniques described here can easily be adapted to device-dependent layers (Physical Layer, Alphabet Layer) as well. Their specification is left to architectures that are designed for specific IDSs.

5.3.1 Data Model and Objects

The main contribution of the reference architecture is to enable the conceptual decomposition of the UI of Design Prototypes into components and to describe the communication between these components. A first step towards this goal is the definition of message types for the definition of interfaces. This section introduces a data

model for the UI layers of Design Prototypes (cf. Figure 5.12) which will be used as a conceptual basis for defining message types in the next section.

Decomposition of Design Prototypes

The structural and behavioural aspects of the reference architecture will be explained with accompanying examples. As explained above, the reference architecture focuses on the *lexical, syntactic, semantic, conversational* and *task* layers of GUI-based Design Prototypes. For demonstration purposes, the following section also includes exemplified states for the device-dependent *physical* and *alphabetic* layers.

We will use two mock-ups from a paper prototype and one Wizard of Oz statement as representations of example states of a single Design Prototype (cf. Figure 5.3):

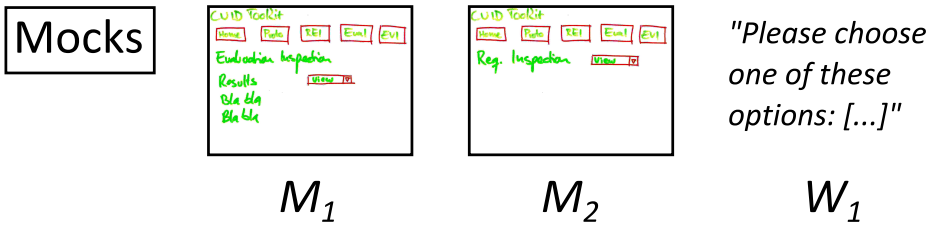

Figure 5.3: Example mocks used in this Chapter.

M_1. This prototype state is represented by a mock-up. It consists of a navigation bar with five buttons, five text fields, and one drop-down list. One of the buttons is supposed to link to the second mock-up, which represents the prototype state M_2.

M_2. This state is again represented by a mock-up. The mock-up shares the navigation bar with the first mock-up. In contrast to the first mock-up, it only includes two text fields and a different drop-down list.

W_1. This state is represented by a Wizard of Oz sentence, i.e. it is a state of a VUI. The sentence is a prompt: "Please choose one of these options: Home, Prototyping, Requirements Inspection, Evaluation".

Although this work focuses on Design Prototypes for GUIs, W_1 has been selected as an example to show that the reference architecture can theoretically be extended to model multi-modal interaction. As mentioned before, the reference architecture does not cover the device-dependent physical and alphabetic layers.

The remainder of this section will describe in detail and by example, how Design Prototypes can be decomposed according to the conceptual model of IDSs. Figure 5.4 provides an overview of the different layers and model elements that will be explained below.

Physical Layer. On the physical layer, the state of the prototype is represented according to the medium which is used for communication. The GUI states (M_1 and M_2) are represented by the colour that is reflected by each given point on the surface of the paper prototype. The VUI state W_1 is represented by the audio frequency

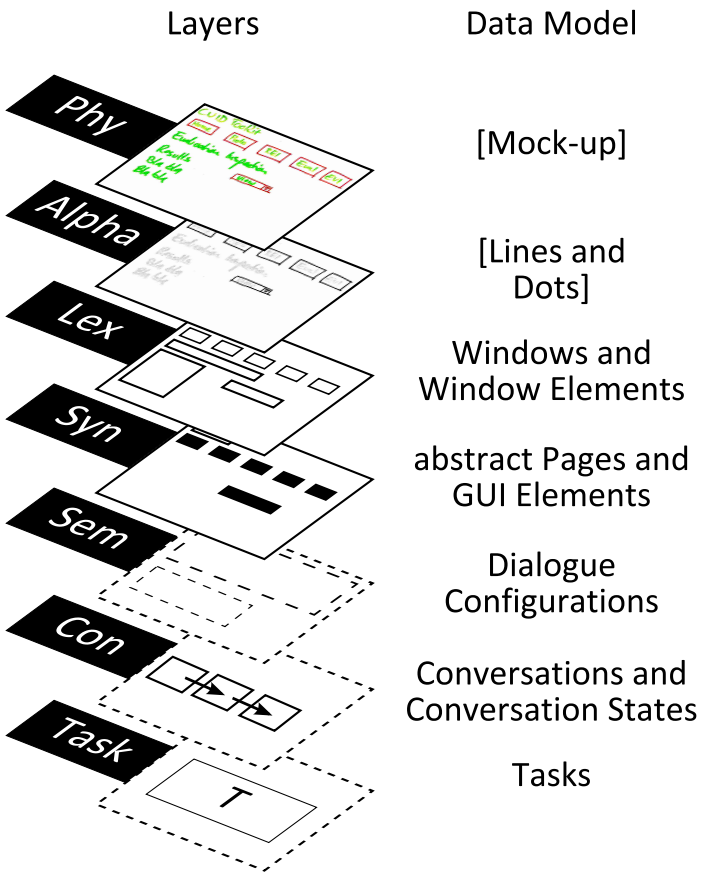

Figure 5.4: Overview of the layers of the conceptual mode of IDSs and how they map to the reference architecture for Design Prototypes.

and the sound pressure level of the sounds that are being emitted from the available speakers (cf. Figure 5.5).

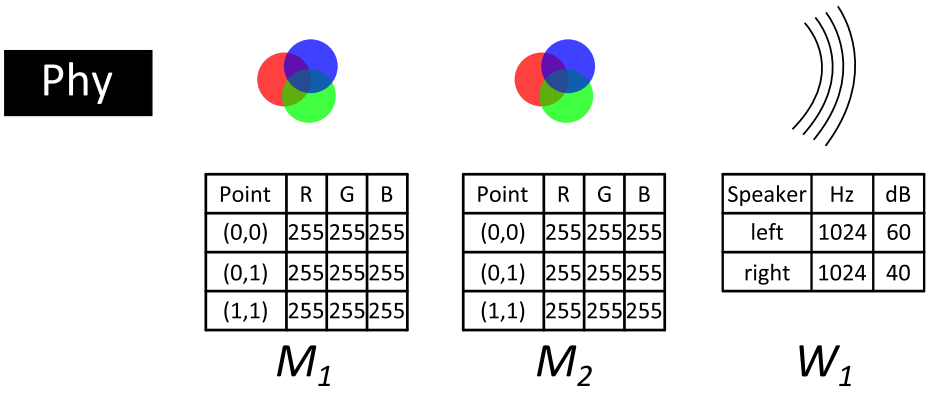

Figure 5.5: Example state on the physical layer.

Alphabetic Layer. On this layer, the GUI states M_1 and M_2 can be described by a set of shapes that are drawn on the mocks (lines and dots). In the example, the two lines $Line_1$ and $Line_2$ (from one of the sketched buttons) are visible both in M_1 and M_2 (cf. Figure 5.6). The dot Dot_1, however, is only visible in M_2. The VUI state is

described by morphemes that are to be uttered. In the example, the morpheme [*p*] is currently being uttered.

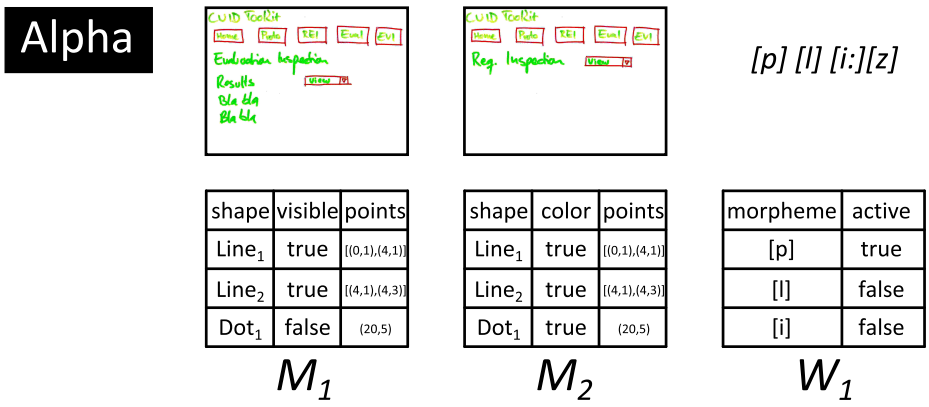

Figure 5.6: Example state on the alphabetic layer.

Lexical Layer. On the lexical layer, GUI state is described by the set of windows and the window elements they contain (e.g. buttons, links and text elements) as well as their visibility. In the example, two buttons from the navigation bar of the Design Prototype (WE_1 and WE_2) are contained in a page that is visible both in M_1 and M_2 (cf. Figure 5.7). The text element WE_3, however, is only visible in M_2. The VUI state W_1 is described by a stack of lexemes and their utterance status *active*.

Generally speaking, for each GUI-based Design Prototype p exists as set of windows W and a set of window elements WE. The fact that window elements are contained in windows, will be described by the relation $R_L : W \times WE$.

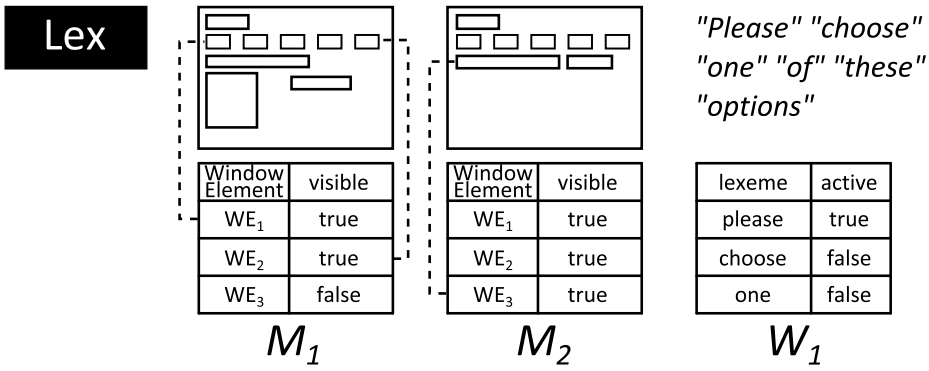

Figure 5.7: Example state on the lexical layer.

Syntax Layer. GUI states on the syntax layer are represented by pages that contain GUI elements, their visibility state and whether they are actionable[1]. In the example, two actionable elements are visible in both M_1 and M_2. The non-actionable element

[1] a GUI element is called *actionable* when the user can interact with the element to case a transition to a different conversational state. For paper prototypes these are elements that can be touched and which cause a different mock to be shown (or the current mock to be adjusted)

E_3 is only visible in M_2. The VUI state W_1 is represented by a sentence that is currently being uttered (cf. Figure 5.8).

Generally speaking, for each GUI-based Design Prototype p, there is a set of pages P and a set of GUI elements E. Some of this elements are *actionable*. The respective subset of E will be denoted by A. GUI elements are contained in pages. This will be denoted by the relation $R_S : P \times E$.

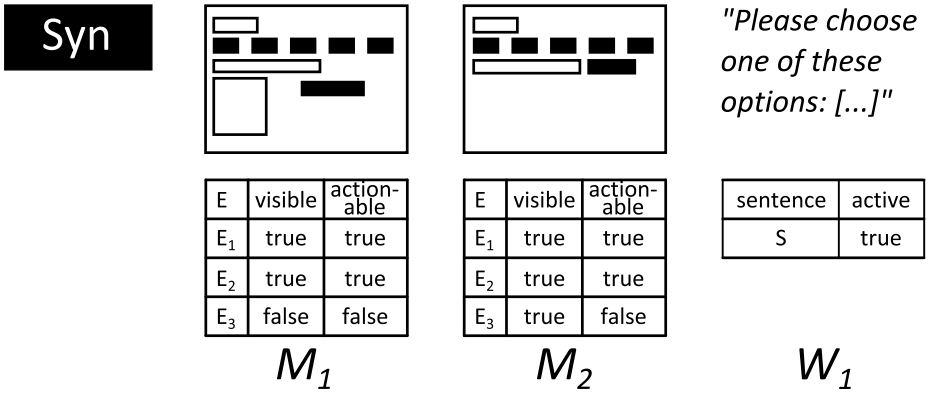

Figure 5.8: Example state on the syntax layer.

Semantic Layer. For Design Prototypes the semantic layer is virtually stateless. The state merely keeps track of which dialogues (of potentially many simultaneous ones) are currently active – irrespective of the underlying modality. In the example, the dialogue that is active in M_1 is D_1, the dialogue that is active in M_2 is D_2 and the one in M_3 is D_3. For each Design Prototype p, there is a set of dialogues. In order to avoid ambiguities with the overloaded term dialogue, this set will be denoted as the set of *Dialogue Configurations* D.

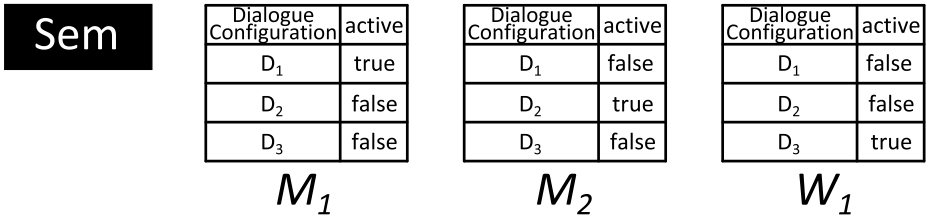

Figure 5.9: Example state on the semantic layer.

Conversation Layer. The conversational layer keeps track of the overall progress of the conversation between user and Design Prototype towards completing a user task. A conversation state can be shared by dialogues of different modalities. In the example, the GUI state M_2 and the VUI state W_1 belong to the same conversation state S_2 (cf. Figure 5.10).

Generally speaking, for each Design Prototype p, there is a set of conversations C.

Task Layer. All three example states share the same task state T_1. Design Prototypes do not always model tasks explicitly. Paper prototypes might come with a

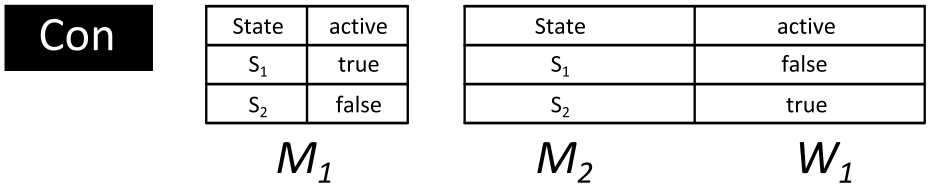

Figure 5.10: Example state on the conversation layer.

storyboard, which groups mocks into tasks. Electronic prototypes might be stored in different files for different user tasks depending on the tool that is used. Generally speaking, for each Design Prototype, there is a set of tasks.

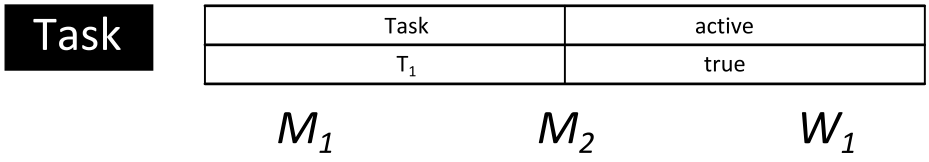

Figure 5.11: Example state on the task layer.

Data Model

Each layer of the reference architecture works on inputs and outputs and uses a *data model* for representing the relations between its elements. Due to the simplicity of the semantics of Design Prototypes most of these models represent static data models. The only exception is the Conversation Layer which is modelled as a state machine. Every Design Prototype p is assigned a Lexical Model Lex_p, a Syntax Model Syn_p, a Semantic Model Sem_p, a Conversational Model Con_p, and a Task Model Tas_p.

Tas_p, Con_p and Sem_p are independent of the modality used. Syn_p and Lex_p are modality-dependent.

Summarizing from the decomposition of Design Prototypes above, the following definitions apply for modality-independent models:

- For each Task Model Tas_p there is a set of Tasks T.
- For each Conversational Model Con_p there is a set of Conversations C.
- For each Semantic Model Sem_p there is a set of Dialogue Configurations D.

The following definitions apply specifically to models of graphical user interfaces:

- The Syntax Model Syn_p is defined as a 4-tuple (P, E, A, R_S):
  - P, the set of pages
  - E, the set of GUI elements
  - $A \subseteq E$, the set of actionable GUI elements
  - $R_S : P \times E$, the relation defining the relationship between pages and GUI elements
- The Lexical Model Lex_p is defined as a 3-tuple (W, WE, R_L):
  - W, the set of windows

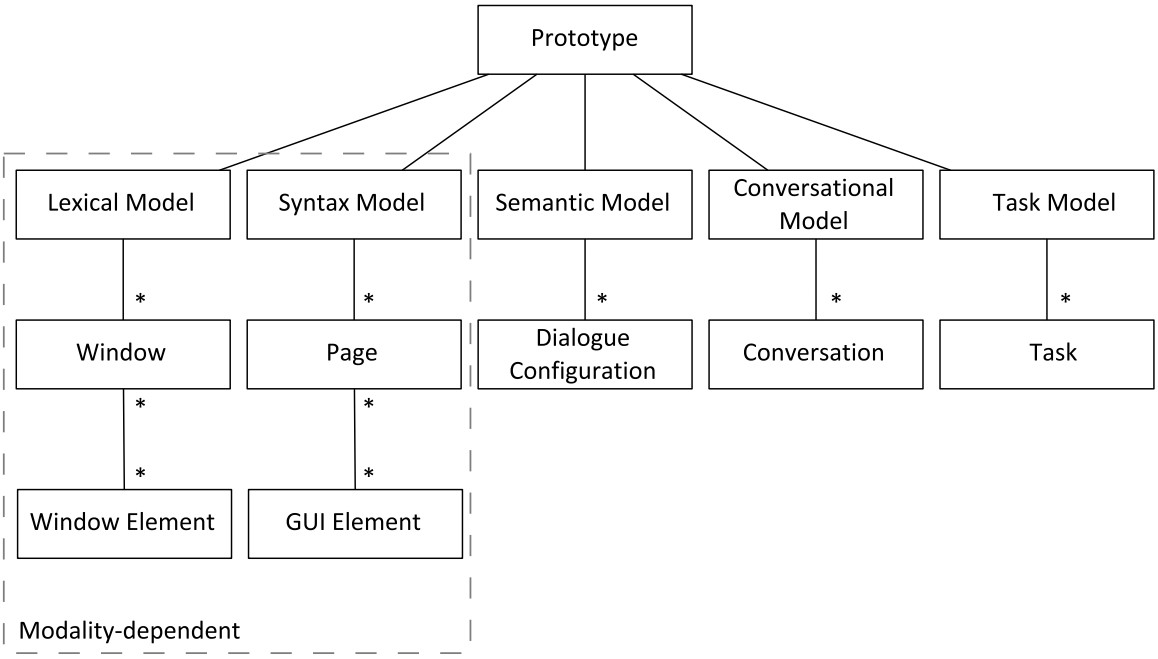

Figure 5.12: The data model reflects the elements of graphical Design Prototypes.

- WE, the set of window elements
- $R_L : W \times WE$, the relation defining the relationship between windows and window elements

Objects

In the following, *Objects* will be used to refer to the set of all element sets of the data model:

$$Objects = \{T; C; D; P; E; A; W; WE\}$$

The elements of *Objects* are named *ObjectTypes*. Each *ObjectType* is a set, whose elements will be called *typed objects*. Each typed object $obj \in ObjectType$ with $ObjectType \in Objects$ has a set of properties $properties.obj$ (see Appendix A for a list of common properties used in this work). For the purpose of this work we assume:

Let $obj \in ObjectType$ with $ObjectType \in Objects$:

$$properties.obj \in \wp\{(n, v) \mid n \in PropertyNames, v \in PropertyValues\}$$

Let $obj \in ObjectType$ with $ObjectType \in Objects$, $p = (n, v) \in properties.obj$ with $n \in PropertyNames$, $v \in PropertyValues$:

$$p.name = n$$
$$p.value = v$$
$$obj.n = v$$

Objects, *ObjectType*s and their properties will be used for the specification of interfaces between layers and the subsequent definition of UI Design Patterns below.

5.3.2 Interface Specification

So far, this Chapter focused on the structural decomposition of Design Prototypes into layers, and the formal representation of the state of a Design Prototype. We will now investigate the behaviour that is represented by a Design Prototype. The description of the behaviour of a Design Prototype will be formalized using interface specifications. The formalization allows the subsequent structured representation of UI Design Patterns in Section 5.4 and the implementation of a prototypical toolkit for detecting UI Design Patterns in Design Prototypes in Chapter 8.

Given that the interface specification has been conceived for Design Prototypes, it is intentionally *underspecified*. An IDS implementation will require significant refinement of this specification after the initial requirements elicitation phase.

Types

The data model can be used as a basis for defining a simple type system for defining interfaces for the layers:

$$Ids = \{WindowElementId; WindowId; ElementId; PageId; ActionId; \\ DialogueConfigurationId; ConversationId; TaskId\}$$

Each $Id \in Ids$ represents a set of unique identifiers for objects of the respective type. f_{Id} provides a type-wise mapping of identifiers to objects:

$$IdFunctions = \{f_{Id} \in Id \mapsto ObjectType \mid Id \in Ids \land ObjectType \in Objects \land \\ \forall x, y \in Id : f_{Id}(x) = f_{Id}(y) \implies x = y\}$$

Behaviour

Figure 5.13 shows an exemplary interaction between a user and a Design Prototype as interpreted using the reference architecture: The interaction starts, when the user touches window element WE_1 (a button) on the mock-up M_1 (not part of the model). The lexical layer receives a message for WE_1 and sends a message for E_1 to the syntax layer. The syntax layer checks that E_1 is an actionable element, and sends a message

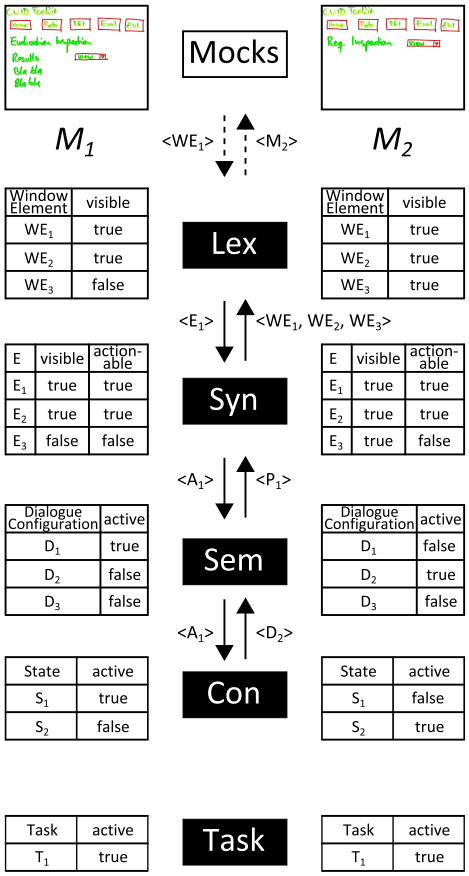

Figure 5.13: Example of an interaction and the respective state transition.

for action A_1 to the semantic layer. The semantic layer forwards this message to the conversation layer. The message does not cause the conversation to be finished, so the conversation layer transitions from state S_1 to state S_2 without sending a message to the task layer. Instead, it sends a message for Dialogue Configuration D_2 to the semantic layer. The semantic layer deactivates dialogue configuration D_1 and activates dialogue configuration D_2. The semantic layer sends a message for page P_1 to the syntax layer. The syntax layer sets page P_1 and all GUI elements that belong to P_1 to *visible*. It sends a message to the lexical layer to make visible all necessary windows and window elements. The lexical layer does so by showing the resulting mock (not part of the model).

The behaviour that is modelled by a Design Prototype can be specified using the type system that has been introduced above and the following auxiliary functions which map input messages to output message streams (Figure 5.14):

$$Functions = \{f_{Lex_I}; f_{Syn_I}; f_{Syn_O}; f_{Sem_I}; f_{Sem_O}; f_{Con_I}; f_{Con_O}; f_{Tas_O}\}$$

- $f_{Lex_I} \in WindowElementId \mapsto ElementId^*$
- $f_{Syn_I} \in ElementId \mapsto ActionId^*$
- $f_{Syn_O} \in PageId \mapsto ElementId^*$
- $f_{Sem_I} \in ActionId \mapsto ActionId^*$

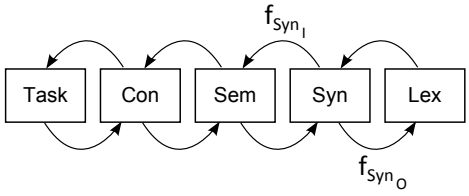

Figure 5.14: Schematic illustration of the auxiliary functions.

- $f_{Sem_O} \in DialogueConfigurationId \mapsto PageId^*$
- $f_{Con_I} \in ActionId \mapsto TaskId^*$
- $f_{Con_O} \in ConversationId \mapsto DialogueConfigurationId^*$
- $f_{Tas_O} \in TaskId \mapsto ConversationId^*$

To reflect the simple semantics of Design Prototypes the functions are defined as:

Let $f \in Functions, id_I \in \text{dom}.f, id_O \in \text{rng}.f, R \in \text{dom}.f \times \text{rng}.f$:

$$f(id_I) = id_O \implies (id_I, id_O) \in R$$

The domain dom.f and range rng.f of f and R can be deduced from the structure of the Design Prototype (see Chapter 6). There are two special cases:

Semantic Layer. As Design Prototypes do not contain differentiated semantics for input actions by the user, the behaviour of the Semantic Layer is modelled by:

Let $id_I \in \text{dom}.f_{Sem_I}, id_O \in \text{rng}.f_{Sem_I}$:

$$f_{Sem_I}(id_I) = id_O \implies id_O = \langle id_I \rangle \tag{5.1}$$

Conversation Layer. As opposed to the other layers, the Conversation Layer is stateful and its output depends both on the input and its current local state. Therefore, the behaviour of each Conversation $c_t \in C$ from the data model is defined as a finite state machine, more precisely a Mealy machine $(\Sigma, \sigma_0, \Omega, I, O, \Delta)$, which can be deduced from a given Design Prototype (Chapter 6):

- Σ, the set of conversational states, one state for each page of the Design Prototype
- $\sigma_0 \in \Sigma$, the initial conversational state
- $\Omega \subseteq \Sigma$, the set of terminal states
- $I \in \text{dom}.f_{Con_I}$, the inputs, one input for each target of actionable elements on the Syntax Layer.
- $O \in \text{rng}.f_{Con_O}$, the outputs, one output for each page on the Syntax Layer.
- $\Delta : \Sigma \times I \times \Sigma \times O$, the state transition relation

Interfaces

Based on the type system and the auxiliary functions an interface specification can be defined that reflects the structure and behaviour that is modelled by a Design

Prototype. The Lexical Layer remains underspecified as its behaviour relies on the Alphabet Layer, which is not specified in the reference architecture.

LexicalLayerInterface

in	$x_1 : WindowId$
out	$y_1 : ElementId$

The Syntax Layer is based on the following semantics: It receives element IDs and page IDs from the Lexical Layer and Semantic Layer, respectively, and maps them to action IDs and a stream of window element IDs, respectively. The latter reflects the fact that the SyntaxLayerInterface is responsible for multiplexing the syntactic concept of a *page* into a set of lexical *window elements*.

SyntaxLayerInterface

in	$x_2 : PageId, y_1 : ElementId$
out	$x_1 : ElementId, y_2 : ActionId$

$f(x_2, y_1) = (x_1, y_2)$
where f so that $\forall e \in ElementId, p \in PageId$:
$f(\langle\rangle, \langle\rangle) = (\langle\rangle, \langle\rangle)$
$f(\langle p\rangle \frown r, \langle e\rangle \frown s) = (o_1, o_2)$ with
$$o_1 = f_{Syn_I}(p) \frown o_3 \wedge o_2 = f_{Syn_O}(e) \frown o_4 \wedge (o_3, o_4) = f(r, s)$$
$f(\langle p\rangle \frown r, \langle\rangle) = (o_1, o_2)$ with
$$o_1 = f_{Syn_I}(p) \frown o_3 \wedge o_2 = o_4 \wedge (o_3, o_4) = f(r, \langle\rangle)$$
$f(\langle\rangle, \langle e\rangle \frown s) = (o_1, o_2)$ with
$$o_1 = o_3 \wedge o_2 = f_{Syn_O}(e) \frown o_4 \wedge (o_3, o_4) = f(\langle\rangle, s)$$

SemanticLayerInterface

in	$x_3 : DialogueConfigurationId, y_2 : ActionId$
out	$x_2 : PageId, y_3 : ActionId$

$f(y_2, x_3) = (y_3, x_2)$
where f so that $\forall a \in ActionId, d \in DialogueConfigurationId$:
$f(\langle\rangle, \langle\rangle) = (\langle\rangle, \langle\rangle)$
$f(\langle a\rangle \frown r, \langle d\rangle \frown s) = (o_1, o_2)$ with
$$o_1 = f_{Sem_I}(a) \frown o_3 \wedge o_2 = f_{Sem_O}(d) \frown o_4 \wedge (o_3, o_4) = f(r, s)$$
$f(\langle a\rangle \frown r, \langle\rangle) = (o_1, o_2)$ with
$$o_1 = f_{Sem_I}(a) \frown o_3 \wedge o_2 = o_4 \wedge (o_3, o_4) = f(r, \langle\rangle)$$
$f(\langle\rangle, \langle d\rangle \frown s) = (o_1, o_2)$ with
$$o_1 = o_3 \wedge o_2 = f_{Sem_O}(d) \frown o_4 \wedge (o_3, o_4) = f(\langle\rangle, s)$$

The semantics of the Design Prototypes discussed in this work are limited to *links* between different *mock-ups*. This is reflected by the specification of the SemanticLayerInterface as it provides no further interpretation for *inputs* from the SyntaxLayerInterface. Instead it simply forwards the input to the ConversationLayerInterface. The output is reflected by the concept of *dialogue configurations*. Instead of being constructed by algorithms during runtime, the syntactical expressions are defined by the static semantic model, which in the case of Design Prototypes is configuration

based: A dialogue configuration is 1:1 mapping of a $DialogueConfigurationId$ to a $PageId$.

The ConversationLayerInterface receives action IDs and conversation IDs as input and returns dialogue configuration IDs and task IDs, respectively, as output. There are two situations, which need to be described: If the Conversation Layer receives a $ConversationId$ message from the Task Layer, the current conversation is cancelled and the current stream of $ActionId$s is disregarded. If the conversation reaches a terminal state in the conversation model the respective $TaskId$ is sent as a message to the Task Layer to indicate that the respective task can be completed.

ConversationLayerInterface

in	$x_4 : ConversationId, y_3 : ActionId$
out	$x_3 : DialogueConfigurationId, y_4 : TaskId$
loc	$currentState \in \Sigma$

$f[\sigma_0](y_3, x_4) = (y_4, x_3)$
where f so that $\forall a \in ActionId, c \in ConversationId, \sigma \in \Sigma$:
$f[currentState](\langle\rangle, \langle\rangle) = (\langle\rangle, \langle\rangle)$
$f[currentState](\langle a\rangle \frown r, \langle c\rangle) = f[currentState](\langle\rangle, \langle c\rangle)$
$f[currentState](\langle a\rangle \frown r, \langle\rangle) = (o_1, o_2)$ with

$$\begin{cases} o_1 = f_{Con_I}(a) \frown o_3 \wedge o_2 = \langle d\rangle \frown o_4 \wedge \\ (o_3, o_4) = f[\sigma](r, s) & \text{if } \exists(currentState, a, \sigma, d) \in \Delta \\ & \wedge \sigma \in \Omega \\ o_1 = o_3 \wedge o_2 = \langle d\rangle \frown o_4 \wedge \\ (o_3, o_4) = f[\sigma](r, s) & \text{if } \exists(currentState, a, \sigma, d) \in \Delta \\ & \wedge \sigma \notin \Omega \end{cases}$$

$f[currentState](\langle\rangle, \langle c\rangle \frown s) = (o_1, o_2)$ with

$$o_1 = o_3 \wedge o_2 = f_{Con_O}(c) \frown o_4 \wedge (o_3, o_4) = f[f_{id}(c).\sigma_0](\langle\rangle, s)$$

The TaskLayerInterface receives task IDs and returns conversation IDs. This reflects the fact that there is a 1:1 mapping between tasks and conversations in the data model. The layer remains underspecified as its behaviour relies on the Application-Layer, who's interface is not part of the reference architecture.

TaskLayerInterface

in	$y_4 : TaskId$
out	$x_4 : ConversationId$

Figure 5.15 summarizes the reference architecture with its layers, data model, interfaces and auxiliary functions.

5.4 A Structured Representation of UI Design Patterns

The reference architecture allows the specification of UI Design Patterns in a structured way as a basis for the application of software engineering methods. While UI

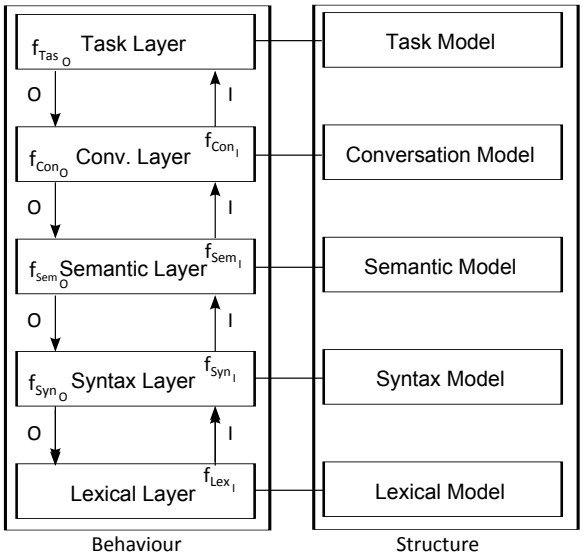

Figure 5.15: Summary of the layers, their data models and interfaces.

Design Patterns used to be described in a prosaic manor, the following formalisms can be used to describe UI Design Patterns as constraints on the structure and behaviour of the Design Prototype. It will be used as a theoretical basis for the analysis of Design Prototypes as part of the case studies in Chapters 6 and 9.

5.4.1 Constraint-based Representation

This section introduces a constraint-based representation for UI Design Patterns. It is accompanied by an explanatory example – the "Global Navigation" design pattern [85]:

> *Using a small section of every page, show a consistent set of links or buttons that take the user to key sections of the site or application.*

Decomposition of UI Design Patterns

The design pattern can be interpreted as constraints on the reference architecture for Design Prototypes (see Figure 5.16). The pattern spans from the lexical layer to the conversation layer.

On the lexical layer, e.g., the pattern requires a set of window elements to be constrained to a certain UI area. This set of window elements will be denoted as $InstanceSet_{WE}$ (these window elements are *instances* of elements of the design pattern). Let $c_{x_1}, c_{y_1}, c_{x_2}, c_{y_2}$ be the constraint values for the properties x_1, y_1, x_2, y_2 of the window elements, so that e.g. $\forall w \in InstanceSet_{WE} : w.x_1 \geq c_{x_1} \wedge w.y_1 \geq c_{y_1} \wedge w.x_2 \leq c_{x_2} \wedge w.y_2 \leq c_{y_2}$. This type of constraint will be denoted as a "data model constraint" below. Similar constraints can be defined to represent the constraints on other layers (see Appendix C).

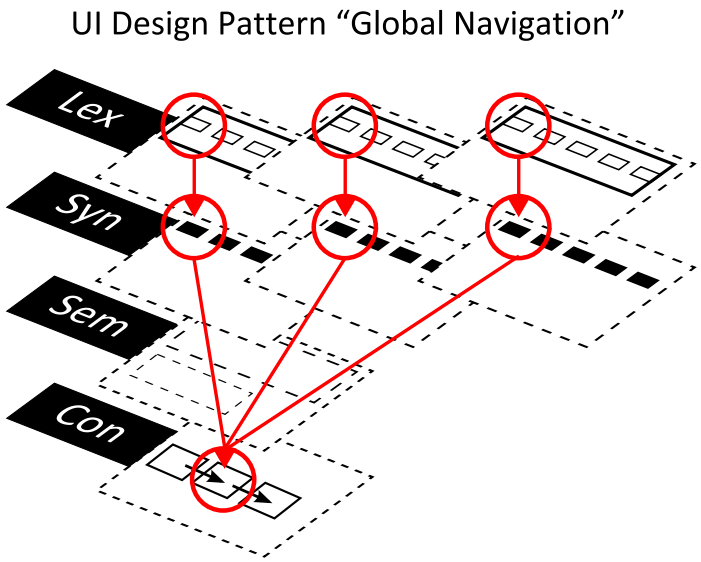

Figure 5.16: UI Design Patterns such as the pattern “Global Navigation” [85] can be represented using the reference architecture for Design Prototypes.

The structured representation of a *pattern* is defined as a set of typed constraints (cf. Figure 5.17). There are three different types of constraints: Behavioural constraints, conversational constraints and data model constraints. They are based on the *data model and types* described above. The constraints make use of three quantifiers: $\forall$, $\exists$, and $\exists_1$. We will use the following notations as abbreviations:

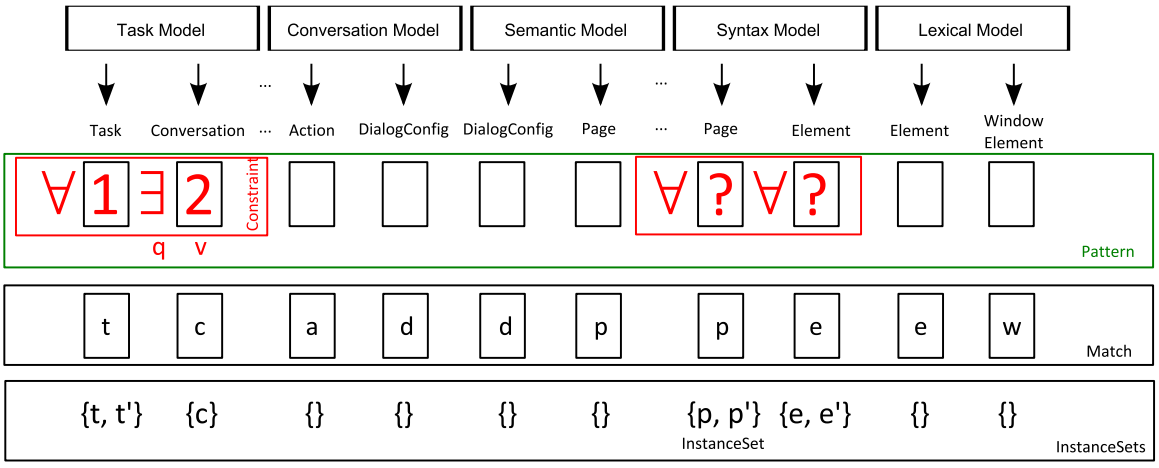

Figure 5.17: Overview of the terms introduced with respect to a structured representation of UI Design Patterns.

$$
\begin{aligned}
Q &= \{\forall, \exists, \exists_1\} \\
V &= \mathbb{N}_0 \cup \{\bot\} \\
\forall q \in Q : q.x.\mathrm{P} &= \begin{cases} \forall x : \mathrm{P}, & \text{if } q \in \{\forall\} \\ \exists x : \mathrm{P}, & \text{if } q \in \{\exists\} \\ \exists_1 x : \mathrm{P}, & \text{if } q \in \{\exists_1\} \end{cases}
\end{aligned}
$$

Behavioural Constraints

A *behavioural constraint* describes constraints with respect to the domain and range of an auxiliary function $f \in Functions$ (see above). It is defined as follows:

Let $f \in Id_1 \mapsto Id_2^*$ with $Id_1, Id_2 \in Ids, Id \in Ids, q_{Id} \in Q, v_{Id} \in V$:

$$
\begin{aligned}
Constraint^f &= (Constraint^f_{Id_1}, Constraint^f_{Id_2}) \\
Constraint^f_{Id} &= (q_{Id}, v_{Id})
\end{aligned}
$$

q_{Id} is called "quantification constraint for type Id" and v_{Id} is called "value size constraint for type Id". The semantics of applying a constraint are described below.

If a Design Prototype implements a behavioural pattern, there will be sets of Ids that instantiate the pattern's constraints for given functions. These sets are called $InstanceSets$:

$$
\begin{aligned}
InstanceSets &\in \wp(\{InstanceSet^f_{Id} \mid Id \in Ids, f \in Functions\}) \\
InstanceSet^f_{Id} &\subseteq Id
\end{aligned}
$$

In order to be able to clarify whether a Design Prototype implements a given pattern, it is first necessary to define the denotation of applying a behavioural $Constraint$ to given $InstanceSet$s:

$$
\begin{aligned}
&[\![Constraint^f.InstanceSets]\!] = \\
&\quad \begin{cases} true, & \text{if } (|InstanceSet^f_{Id_1}| = v_{Id_1} \vee v_{Id_1} \in \{\bot\}) \\ & \wedge(|InstanceSet^f_{Id_2}| = v_{Id_2} \vee v_{Id_2} \in \{\bot\}) \\ & \wedge q_{Id_1}.(val_1 \in InstanceSet^f_{Id_1}).(q_{Id_2}.(val_2 \in InstanceSet^f_{Id_2}). \\ & (\exists i \in \mathbb{N} : val_2 = f(val_1)(i))) \\ false, & \text{otherwise.} \end{cases}
\end{aligned}
$$

This definition models a logic predicate of the form:

$\boxed{q_{Id_1}} val_1 \in InstanceSet^f_{Id_1} \boxed{q_{Id_2}} val_2 \in InstanceSet^f_{Id_2} \wedge (|InstanceSet^f_{Id_1}| = v_{Id_1} \vee v_{Id_1} \in \{\bot\}) \wedge (|InstanceSet^f_{Id_2}| = v_{Id_2} \vee v_{Id_2} \in \{\bot\}) \exists i \in \mathbb{N} : val_2 = f(val_1)(i)$

Conversational Constraints

A *conversational constraint* describes constraints on the state transition relation $\Delta \in \Sigma \times I \times \Sigma \times O$ (see above). Conversational constraints are defined as follows:

Let $\Sigma_1 = \Sigma_2 = \Sigma, \Delta \in \Sigma_1 \times I \times \Sigma_2 \times O, a \in \{\Sigma_1; I; \Sigma_2; O\}, q_a \in Q, v_a \in V$:

$$Constraint^\Delta = (Constraint^\Delta_{\Sigma_1}, Constraint^\Delta_I, Constraint^\Delta_{\Sigma_2}, Constraint^\Delta_O)$$
$$Constraint^\Delta_a = (q_a, v_a)$$

q_a is called "quantification constraint for relation argument a" and v_a is called "value size constraint for relation argument a". The respective *InstanceSets* are defined as:

$$InstanceSets \in \wp(\{InstanceSet^\Delta_a \mid a \in \{\Sigma_1; I; \Sigma_2; O\}\})$$
$$InstanceSet^\Delta_a \subseteq a$$

The denotation of applying a conversational *Constraint* to given *InstanceSets* is defined as:

$$[\![Constraint^\Delta.InstanceSets]\!] =$$
$$\begin{cases} true, & \text{if } (|InstanceSet^\Delta_I| = v_I \vee v_I \in \{\bot\}) \\ & \wedge(|InstanceSet^\Delta_{\Sigma_2}| = v_{\Sigma_2} \vee v_{\Sigma_2} \in \{\bot\}) \\ & \wedge(|InstanceSet^\Delta_O| = v_O \vee v_O \in \{\bot\}) \\ & \wedge(|InstanceSet^\Delta_{\Sigma_1}| = v_{\Sigma_1} \vee v_{\Sigma_1} \in \{\bot\}) \\ & \wedge q_{\Sigma_1}.(val_1 \in InstanceSet^\Delta_{\Sigma_1}).(q_I.(val_2 \in InstanceSet^\Delta_I). \\ & (q_{\Sigma_2}.(val_3 \in InstanceSet^\Delta_{\Sigma_2}).(q_O.(val_4 \in InstanceSet^\Delta_O). \\ & ((val_1, val_2, val_3, val_4) \in \Delta)))) \\ false, & \text{otherwise.} \end{cases}$$

Data Model Constraints

Data model constraints describe constraints on the data model (see above) by defining constraints with respect to a typed object $obj \in ObjectType$ with $ObjectType \in Objects$ and its properties properties.obj:

Let $f_{ObjectType} \in IdFunctions, ObjectType \in Objects, obj \in ObjectType, q \in Q, v \in V, p \in properties.obj, c \in \{<, \leq, =, \geq, >\}, cv \in V$:

$$Constraint^{f_{ObjectType}} = (q, v, p, c, cv)$$

The respective $InstanceSets$ are defined as:

$$\begin{aligned} InstanceSets \in \wp(\{InstanceSet^{f_{ObjectType}} \mid ObjectType \in Objects, \\ f_{ObjectType} \in IdFunctions\}) \\ InstanceSet^{f_{ObjectType}} \subseteq ObjectType \end{aligned}$$

The denotation of applying a data model $Constraint$ to its related $InstanceSet$s is given as:

Let $o \in IdFunctions$:

$$[\![Constraint^o.InstanceSets]\!] = \begin{cases} true, & \text{if } (|InstanceSet^o| = v \vee v \in \{\bot\}) \wedge q.(obj \in InstanceSet^o). \\ & ([\![p.value \diamond cv]\!]) \\ false, & \text{otherwise.} \end{cases}$$

Let $v, cv \in V, \diamond \in \{<, \leq, =, \geq, >\}$:

$$[\![val \diamond cv]\!] = \begin{cases} true, & \text{if } \diamond \in \{<\} \wedge val < cv \\ true, & \text{if } \diamond \in \{\leq\} \wedge val \leq cv \\ true, & \ldots \\ true, & val \in \{\bot\} \vee cv \in \{\bot\} \\ false, & \text{otherwise.} \end{cases}$$

Patterns

With this basic formalism a $Pattern$ can be constructed by defining a set of $Constraint$s:

$$Pattern \subseteq \wp(Constraint^f \cup Constraint^\Delta \cup Constraint^o)$$

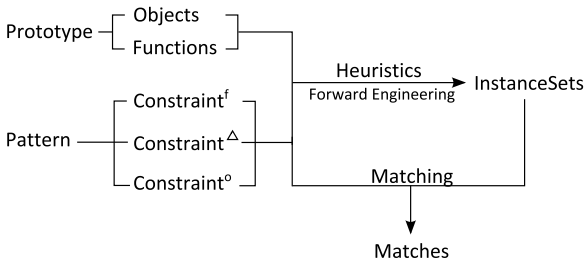

Figure 5.18: Using constraints for matching of prototypes against patterns.

5.4.2 Matching

Each *Pattern* defines constraints concerning the size of the value sets that should be matched, and the number of elements in the set that need to match in order for the pattern to be matched.

Matching is defined by a *match* function:

Let $g \in Functions \cup IdFunctions \cup \{\Delta\}, is \in InstanceSets, p \in Pattern$:

$$match : InstanceSets \times Pattern \mapsto true, false$$
$$match(is, p) = \bigwedge_{Constraint^g \in p} [\![Constraint^g.is]\!] \tag{5.2}$$

Limitations

The structured representation for UI Design Patterns that has been introduced in this section focuses on the structural aspects of Design Prototype. It cannot be used for describing timing constraints or arbitrary logical predicates. Chapter 10 lists potential extensions to this approach.

5.5 Summary

This chapter introduced structured representations for Design Prototypes and UI Design Patterns. Figure 5.18 illustrates the relationship between the constructs introduced above and how they can be used to match parts of Design Prototypes (*InstanceSets*) against the structured representation of a given design pattern. This approach allows to decide whether a given Design Prototype incorporates a given design pattern.

454 - average number of GUI elements in a Design Prototype

study result

6 Study: Implicit Requirements in Design Prototypes

The structured representation of Design Prototypes and UI Design Patterns (cf. Chapter 5) can be used to identify patterns in structured representations of Design Prototypes. In 2010, a confirmatory case study was conducted together with Pidoco GmbH, Berlin, with the goal of identifying UI Design Patterns in Electronic Prototypes that were created in real software engineering projects. The results confirm the existence of implicit requirements in the form of UI Design Patterns in real Design Prototypes and the ability of the approach introduced in Chapter 5 to represent both the prototypes as well as the contained patterns in a structured way. As mentioned before, this study report is structured according to Runeson and Höst [70].

6.1 Introduction

Electronic Prototyping is a prototyping method which is already used in professional software engineering projects. The study partner Pidoco GmbH offers a subscription- and web-based service called "Pidoco" which allows for collaborative Electronic Prototyping. This study examines the structure and the implicit requirements contained in Electronic Prototypes created with Pidoco. It was conducted on a post mortem basis, i.e., it is based on projects that have been conducted before the beginning of this study.

Problem Statement

The requirements gathered with electronic prototypes touch all layers of an IDS (see Section 3.2.3). Chapter 5 argues that a structured representation for Design Prototypes can help make the requirements contained in Design Prototypes become explicit. Specifically, it is assumed that it makes identifying and tracing requirements such as UI Design Patterns easier.

Research Objective

This study was conducted in order to

1. examine how realistic Design Prototypes and their implicit requirements can be represented in a more structured way

2. confirm the preliminary hypothesis from the previous exploratory case study (cf. Chapter 4) by showing that realistic Design Prototypes contain implicit requirements across all layers of the model defined in Section 2.3 in the form of UI Design Patterns.

Study Partner

The study partner was selected with the goal of getting access to archival data with Design Prototypes from different professional projects:

Pidoco GmbH. Pidoco focuses on early iterations of the UI Design process from first sketches (low-fidelity prototypes) to clickable mock-ups (high-fidelity prototypes). At the time of the study, it comprised the following features: Sketching GUIs (freehand, geometry tools, graphics import), specifying simple website wireframes (click paths), click-through testing of wireframes, support for reviewer comments (annotations), and a simple versioning system that supports basic iterative design. The resulting artefacts are electronic prototypes as described in Section 3.2.1. Please refer to Section 4.1 for a general introduction of Pidoco GmbH.

6.2 Case Study Design

Research Questions

Section 5.4.2 introduced a mechanism for confirming the existence of a given design pattern in a Design Prototype. As part of this case study it is applied to Design Prototypes to examine the second top level research question of this work:

> **RQ 2** How can realistic Design Prototypes and their implicit requirements be represented in a more structured way?

As the mechanism is based on a structured representation both for UI Design Patterns as well as Design Prototypes (cf. Figure 5.18), this study involved three specific research questions:

RQ 2.1 *What does a mapping from realistic Design Prototypes to a structured representation using the reference architecture from Section 5.3 look like?*

Can the structured representation introduced in Chapter 5 be used to represent professional prototypes in a structured way? This is of interest as it is the basis for applying more formal methods of software engineering such as requirements tracing and more specifically to apply the mechanism for identifying UI Design Patterns in Design Prototypes from Section 5.4.2.

RQ 2.2 *What does a mapping from realistic UI Design Patterns to a structured representation using constraints from Section 5.4 look like?*

If professional Design Prototypes (in this case electronic prototypes) contain implicit requirements, can the structured representation introduced in Chapter 5 be used to

represent the UI Design Patterns in a structured way? Given that there are numerous different UI Design Patterns, this study focuses on four UI Design Patterns found in Tidwell 2005 [85]: *Wizard* [85, p. 42], *Global Navigation* [85, p. 66], *Pyramid* [85, p. 71] and *Sequence Map* [85, p. 76]. The patterns were selected to collectively provide full coverage of all layers of the model in Section 2.3.

RQ 2.3 *How many Design Prototypes from professional software engineering projects contain implicit requirements?*

As the previous work has been mainly of academic character, it is of interest to understand the extent to which implicit requirements are actually contained in professional Design Prototypes. This can be achieved by applying the mechanism described in Section 5.4.2 using the structured representations that are the results of the analysis for RQs 2.1 and 2.2.

Case and Subjects Selection

The case under investigation (Pidoco) is a professional online service for creating electronic prototypes. The subjects that have been selected for the study are 49 electronic prototypes from 10 different professional software engineering projects. As the case study focuses on post mortem data of 3rd party customer projects, the subjects have been selected based on the legal ability of the study partner to contribute the subjects to the study. Therefore, the subjects have been drawn from projects which had been created under licensing terms which allow Pidoco GmbH to gain insights from those customer projects for future product development.

Data Collection Procedures

The study is based on a third-degree method, the analysis of archival data: The datasets for the 49 electronic prototypes come from Pidoco's project repository. Before being contributed to the study, the data has been anonymized, i.e. all human readable text and images have been removed. What remained is the essence of the electronic prototype in terms of its structure and behaviour. The prototypes have been provided to the study as XML files.

Analysis Procedures

The analysis was split up into four phases and nine steps:

- Phase 1 addresses research questions **RQ 2.1** and **RQ 2.2** by (manually or heuristically) defining a mapping from UI Design Patterns and Design Prototypes to their respective structured representation.
- Phase 2 addresses research questions **RQ 2.1** and **RQ 2.2** by applying the mappings from Phase 1 to a set of realistic Design Prototypes.
- Phase 3 addresses research question **RQ 2.3** by using the structured representations from Phase 2 for identifying UI Design Patterns in the given set of Design Prototypes.

- Phase 4 includes the validity procedures put in place to support the results from Phase 3.

Phase 1: Preparation

Step 1: The formalism introduced in Section 5.4 was used to define structured representations for four UI Design Patterns found in Tidwell 2005 [85]: *Wizard* [85, p. 42], *Global Navigation* [85, p. 66], *Pyramid* [85, p. 71] and *Sequence Map* [85, p. 76].

Step 2: Although heuristics are not an intended contribution of this work, the definition of heuristics for identifying design patterns in Design Prototypes was necessary for an automated identification of the selected UI Design Patterns.

Step 3: In order to conduct the subsequent analyses a software tool has been implemented for processing the study data.

Phase 2: Pre-Processing

Step 4: Using the software tool, the Electronic Prototypes where converted into structured representations.

Phase 3: Pattern Identification

Step 5: Potential *InstanceSets* of UI Design Patterns where identified by applying the heuristics to the structured representations of the Electronic Prototypes.

Step 6: The constrained-based representation of the patterns was matched to the heuristically identified potential *InstanceSets* of UI Design Patterns as described in Section 5.4.

Step 7: The results were logged to comma-separated value files (CSV).

Phase 4: Post-Processing

Step 8: The CSV files were used to select random samples from the identified UI Design Patterns. Documents were created containing the identifier of the original XML file, a visual representation of the structured representation of the Design Prototype, a list of identified patterns and a visual representation of the identified *InstanceSets*.

Step 9: The result samples from Step 8 were reviewed by usability experts from Pidoco in order to confirm the identified UI Design Patterns.

Validity Procedures

The validity of the results is supported by two aspects. First, the study was conducted on *archival data*, i.e. the units of analysis were not affected by the goals of the study or the experimenter. Second, the results were *triangulated* by performing manual reviews with usability experts after the study data was analyzed. Usability experts at Pidoco GmbH were provided with result samples and were asked to indicate whether they agree with the UI Design Patterns identified during the analysis.

6.3 Results

The case study confirms that it is *feasible to deduce a structured representation from real Design Prototypes and to define UI Design Patterns using a structured representation.* Furthermore, it confirms that Design Prototypes contain implicit requirements in the form of UI Design Patterns.

6.3.1 Case and Subject Description

49 electronic prototypes have been analyzed. The amount of data contained in these prototypes can be conveyed by providing some key figures: There were 514 wireframes with 2,720 transitions and 22,250 GUI elements, which translates to an average of 10 wireframes, 56 transitions and 454 GUI elements per prototype. Figure 6.1 provides an overview of the distribution of GUI elements and transitions among the subjects.

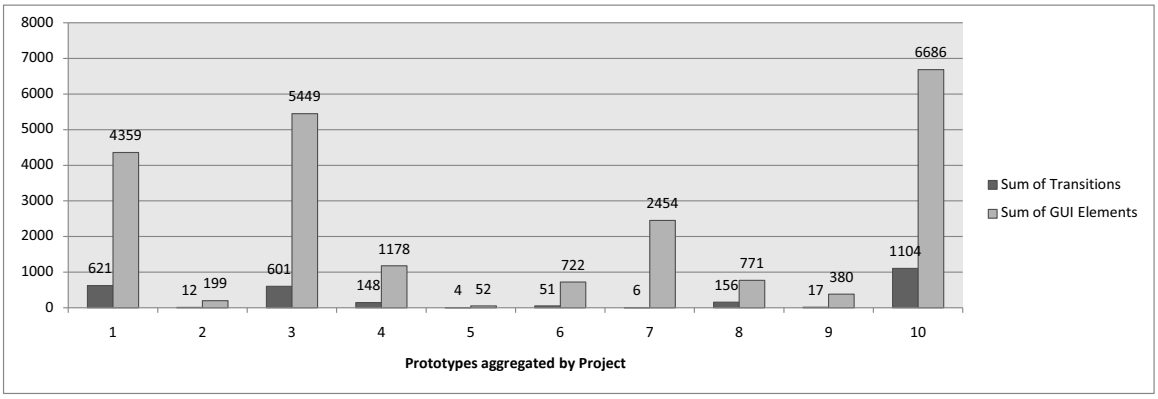

Figure 6.1: Design Prototypes are much more complex than their informal character suggests.

6.3.2 What does a mapping from realistic Design Prototypes to a structured representation using the reference architecture from Section 5.3 look like? (RQ 2.1)

As part of the study, the 49 electronic prototypes were translated to a structured representation. This was achieved by defining a mapping between the original data model of the electronic prototypes and the structured representation in Chapter 5.

Data model and objects. The original data model is based on a set of typed objects (*PidocoObjects*), which contains typed objects of the types *Prototype*, *Page*, *Layer* and *Element* as illustrated in Figure 6.2. The data model also defines a containment hierarchy, which can be modelled by introducing a property for all $x \in PidocoObjects : components.x$, and a number of functions:

Let $PidocoObjects = Prototype \cup Page \cup Layer \cup Element, x \in PidocoObjects, y \in PidocoObjects$:

$$
\begin{aligned}
contains &\in PidocoType \times PidocoType \mapsto \{true, false\} \\
contains(x, y) = true &\implies y \in components.x \\
linkable &\in PidocoType \mapsto \{true, false\} \\
linkable(x) = true &\implies \exists l, t \in properties.x : \\
l.name = \text{“linkable”} &\wedge l.value = true \\
\wedge t.name = \text{“target”} &\wedge t.value \neq \emptyset
\end{aligned}
$$

Mapping functions. The mappings were defined using the mapping functions shown in Collection 6.1 and Collection 6.2, which were then applied to each *Prototype*.

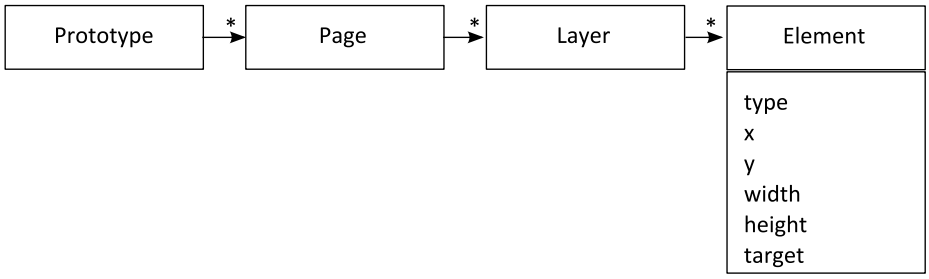

Figure 6.2: Original data model of pidoco prototypes.

Each mapping function $m_{Pr \mapsto T}, m_{Pr \mapsto C}, m_{Pa \mapsto D}, m_{Pa \mapsto P}, m_{Pa \mapsto W}$ defines a bijective mapping between objects of type *Prototype* and *Page* from Pidoco's data model to objects of type *task, conversation, dialogue, page* and *window* (T, C, D, P, W) in the data model of the reference architecture (cf. Collection 6.1). This mapping reflects the fact that prototypes and pages in Pidoco's data model represent concepts across several layers of the reference architecture: objects of type *Prototype* represent user tasks and conversation models, objects of type *Page* represent dialogues, pages and windows.

$m_{Pr \mapsto T} \in Prototype \mapsto T$	$m_{Pr \mapsto T}$ bijective
$m_{Pr \mapsto C} \in Prototype \mapsto C$	$m_{Pr \mapsto C}$ bijective
$m_{Pa \mapsto D} \in Page \mapsto D$	$m_{Pa \mapsto D}$ bijective
$m_{Pa \mapsto P} \in Page \mapsto P$	$m_{Pa \mapsto P}$ bijective
$m_{Pa \mapsto W} \in Page \mapsto W$	$m_{Pa \mapsto W}$ bijective

Collection 6.1: Mapping of Prototype and Page elements from the prototype data model to a structured representation

Pidoco's data model also contains the concept of *layers*. Layers are an implementation detail in this model that allow for the modularization and re-use of Design Prototype components. Each layer contains objects of type *Element*. Three mapping functions $m_{E\times L\times Pa\mapsto WE}, m_{E\times L\times Pa\mapsto A}, m_{E\times L\times Pa\mapsto E}$ are introduced to map objects of type *Element* to the conceptual model, namely to objects of type *GUI Element, actionable GUI Element,* and *Window Element* (E, A, WE) (cf. Collection 6.2).

The relations between pages and GUI elements (R_S) and the relations between windows and window elements (R_L) are deduced from the containment hierarchy of the elements, layers and pages in the Pidoco model. Both relations are equivalent due to the simplicity of the given Design Prototype model.

Types. The necessary types were defined using $IdFunctions$ (cf. Chapter 5):

$$\text{Let } ObjectType \in Objects, Id \in Ids, (f_{Id} \in ObjectType \mapsto Id) \in IdFunctions :$$
$$\forall obj \in ObjectType : f_{Id}(obj) = id \implies id \in Id$$

Behaviour. The behaviour of the interfaces is defined by the auxiliary functions as described in Chapter 5. In order to map the examined Design Prototypes to a structured representation the auxiliary functions were defined as presented in Collection 6.3: f_{Lex_I} reflects the structural equivalence between GUI elements on the syntax layer and window elements on the lexical layer in the Pidoco model. f_{Syn_I} reflects that there is an atomic action on the semantic layer for each actionable element on the syntax layer. This is different from the opposite direction, f_{Syn_O}, where one page on the syntax layer might be represented by multiple window elements on the lexical layer. f_{Sem_O} is a bijective mapping between dialogue configurations from the conversation layer and pages from the syntax layer. f_{Con_I} maps an action to a task, and f_{Tas_O} maps a task to a conversation.

The auxiliary function f_{Sem_I} was already defined in Function 5.1. As a prerequisite for f_{Con_O}, the model $(\Sigma, \sigma_0, \Omega, I, O, \Delta)$ of a conversation needs to be defined. An electronic prototype represents a single conversation with its possible states. Therefore, $(\Sigma, \sigma_0, \Omega, I, O, \Delta)$ is defined by

$\Sigma = PageId$, $I = ActionId$, $O = DialogueConfigurationId$ and the state transition relation Δ:

$$\begin{aligned}
&\forall i \in I, o \in O, s_1 \in \Sigma, s_2 \in \Sigma,\\
&pa_1, pa_2 \in Page, l \in Layer, el \in Element, p_1, p_2 \in P :\\
&contains(l, el) = true \wedge contains(p_1, l) = true\\
&\wedge linkable(el) = true \wedge el.target = p_2\\
&\wedge m_{Pa\mapsto P}(pa_1) = p_1 \wedge m_{Pa\mapsto P}(pa_2) = p_2\\
&\wedge m_{E\times L\times P\mapsto A}(el, l, pa_1) = i \wedge m_{P\mapsto D}(pa_2) = o\\
&\iff (s_1, i, s_2, o) \in \Delta
\end{aligned}$$

$$m_{E\times L\times Pa\mapsto WE} \in Element \times Layer \times Page \mapsto WE :$$
$$\forall e_1, e_2 \in Element, l_1, l_2 \in Layer, p_1, p_2 \in Page, we \in WE \text{ with}$$
$$contains(l_1, e_1) = true \wedge contains(p_1, l_1) = true$$
$$\wedge we = m_{E\times L\times Pa\mapsto WE}(e_1, l_1, p_1) = m_{E\times L\times Pa\mapsto WE}(e_2, l_2, p_2)$$
$$\implies e_1 = e_2 \wedge l_1 = l_2 \wedge p_1 = p_2$$

$$m_{E\times L\times Pa\mapsto A} \in Element \times Layer \times Page \mapsto A :$$
$$\forall e_1, e_2 \in Element, l_1, l_2 \in Layer, p_1, p_2 \in Page, a \in A \text{ with}$$
$$contains(l_1, e_1) = true \wedge contains(p_1, l_1) = true \wedge linkable(e_1) = true$$
$$\wedge a = m_{E\times L\times Pa\mapsto A}(e_1, l_1, p_1) = m_{E\times L\times Pa\mapsto A}(e_2, l_2, p_2)$$
$$\implies e_1 = e_2 \wedge l_1 = l_2 \wedge p_1 = p_2$$

$$m_{E\times L\times Pa\mapsto E} \in Element \times Layer \times Page \mapsto E :$$
$$\forall e_1, e_2 \in Element, l_1, l_2 \in Layer, p_1, p_2 \in Page, e \in E \text{ with}$$
$$contains(l_1, e_1) = true \wedge contains(p_1, l_1) = true$$
$$\wedge e = m_{E\times L\times Pa\mapsto E}(e_1, l_1, p_1) = m_{E\times L\times Pa\mapsto E}(e_2, l_2, p_2)$$
$$\implies e_1 = e_2 \wedge l_1 = l_2 \wedge p_1 = p_2$$

The structural relations R_S and R_L are defined as:

$$\forall e_p \in Element, l_p \in Layer, p_p \in Page, e \in E, p \in P :$$
$$contains(l_p, e_p) = true \wedge contains(p_p, l_p) = true$$
$$\wedge m_{Pa\mapsto P}(p_p) = p \wedge m_{E\times L\times P\mapsto E}(e_p, l_p, p_p) = e$$
$$\iff (p, e) \in R_S$$

$$\forall e_p \in Element, l_p \in Layer, p_p \in Page, we \in WE, w \in W :$$
$$contains(l_p, e_p) = true \wedge contains(p_p, l_p) = true$$
$$\wedge m_{Pa\mapsto W}(p_p) = w \wedge m_{E\times L\times P\mapsto WE}(e_p, l_p, p_p) = we$$
$$\iff (p, e) \in R_L$$

$$Objects = \{T; C; D; P; E; A; W; WE\}$$

Collection 6.2: Mapping of Layer elements from the prototype data model to a structured representation

Ω is defined as $\forall s_1 \in \Sigma, \nexists i \in I, o \in O, s_2 \in \Sigma : (s_1, i, s_2, o) \in \Delta \implies s_1 \in \Omega$. As the initial conversational state σ_0 is not modelled by the original archival data, it has been heuristically identified by performing a breadth-first uni-weight minimum-spanning

$$\forall el \in Element, l \in Layer, p \in Page, e \in E, we \in WE,$$
$$id_e \in Id_1, id_{we} \in Id_2, Id_1, Id_2 \in Ids :$$
$$m_{E\times L\times Pa\mapsto E}(el, l, p) = e \wedge m_{E\times L\times Pa\mapsto WE}(el, l, p) = we$$
$$\wedge f_{Id_2}(id_{we}) = we \wedge f_{Id_1}(id_e) = e$$
$$\implies f_{Lex_I}(id_{we}) = \langle id_e \rangle$$

$$\forall el \in Element, l \in Layer, p \in Page, a \in A,$$
$$id_a \in Id_1, id_e \in Id_2, Id_1, Id_2 \in Ids :$$
$$linkable(el) = true \wedge el.target = p \wedge m_{E\times L\times Pa\mapsto A}(el, l, p) = a$$
$$\wedge f_{Id_2}(id_e) = e \wedge f_{Id_1}(id_a) = a$$
$$\implies f_{Syn_I}(id_e) = \langle id_a \rangle$$
$$\forall el \in Element, l \in Layer, pa \in Page, p \in P, w \in W, we \in WE,$$
$$id_p \in Id_1, id_{we} \in Id_2, Id_1, Id_2 \in Ids : contains(pa, l) \wedge contains(l, el)$$
$$\wedge m_{E\times L\times Pa\mapsto WE}(el, l, pa) = we \wedge m_{Pa\mapsto P}(pa) = p \wedge m_{Pa\mapsto W}(pa) = w$$
$$\wedge f_{Id_1}(id_p) = p \wedge f_{Id_2}(id_{we}) = we$$
$$\implies \exists i \in \mathbb{N} : f_{Syn_O}(id_p)(i) = id_{we}$$

$$\forall d \in D, pa \in Page, p \in P, id_d \in Id_1, id_p \in Id_2, Id_1, Id_2 \in Ids :$$
$$m_{P\mapsto D}(pa) = d \wedge m_{Pa\mapsto P}(pa) = p \wedge f_{Id_1}(id_d) = d \wedge f_{Id_2}(id_p) = p$$
$$\implies f_{Sem_O}(id_d) = \langle id_p \rangle$$

$$\forall pr \in Prototype, pa \in Page, l \in Layer, el \in Element, a \in A, t \in T,$$
$$id_a \in Id_1, id_t \in Id_2, Id_1, Id_2 \in Ids :$$
$$contains(pr, pa) = true \wedge contains(pa, l) = true \wedge contains(l, el) = true$$
$$linkable(el) = true \wedge el.target = pa \wedge m_{E\times L\times Pa\mapsto A}(el, l, pa) = a$$
$$\wedge m_{Pr\mapsto T}(pr) = t \wedge f_{Id_1}(id_a) = a \wedge f_{Id_2}(id_t) = t$$
$$\implies f_{Con_I}(id_a) = \langle id_t \rangle$$

$$\forall p \in Prototype, c \in C, t \in T, id_c \in Id_1, id_t \in Id_2, Id_1, Id_2 \in Ids :$$
$$m_{Pr\mapsto C}(p) = c \wedge m_{Pr\mapsto T}(p) = t \wedge f_{Id_1}(id_c) = c \wedge f_{Id_2}(id_t) = t$$
$$\implies f_{Tas_O}(id_t) = \langle id_c \rangle$$

Collection 6.3: Mapping of behavioural parts of the prototype model to auxiliary functions of the structured representation

tree search on the graph, which is defined by Δ. f_{Con_O} can now be defined for each Design Prototype as:

$$\begin{aligned} \forall pr \in Prototype, pa \in Page, c \in C, d \in D, \sigma_0 : \\ \wedge\, m_{Pr \mapsto C}(pr) = c \wedge m_{Pa \mapsto D}(pa) = d \\ \wedge\, contains(pr, pa) = true \wedge \sigma_0 = d \\ \wedge\, f_{Id_1}(id_c) = c \wedge f_{Id_2}(id_d) = d \\ \implies \quad f_{Con_O}(id_c) = \langle id_d \rangle \end{aligned}$$

Figure 6.3 illustrates a state transition relation that was extracted from one of the prototypes in the study. It provides an impression of the complexity of interaction that can be contained even in early Electronic Prototypes.

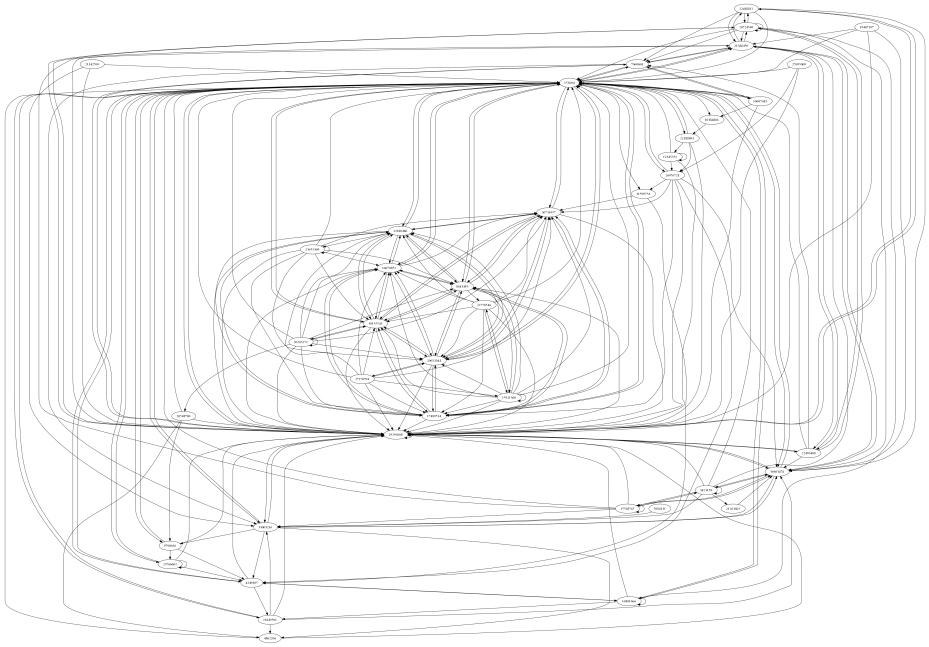

Figure 6.3: Simplified example conversation state transition graph without transition labels.

Using the mappings in Collections 6.1, 6.2 and 6.3 the 49 electronic prototypes could be translated to 49 prototypes with a structured representation.

6.3.3 What does a mapping from realistic UI Design Patterns to a structured representation using constraints from Section 5.4 look like? (RQ 2.2)

As part of the study, four design patterns were translated to a structured representation, namely the *Global Navigation* pattern, the *Pyramid Pattern*, the *Wizard Pattern* and the *Sequence Map Pattern* [85]. This was achieved by mapping the prosaic descriptions of UI Design Patterns to the types of constraints introduced in Section 5.4.

Statement	**Layer**	*Constraint* **and** *InstanceSets*
"Using a small section of every page, show a consistent set of links or buttons that take the user to key sections of the site or application."	Conversation Layer	$Constraint^{\Delta} = \{Constraint^{\Delta}_{\Sigma_1}, Constraint^{\Delta}_{I}, Constraint^{\Delta}_{\Sigma_2}, Constraint^{\Delta}_{O}\}$ $Constraint^{\Delta}_{\Sigma_1} = (\forall, \lvert InstanceSet^{\Delta}_{\Sigma_1}\rvert)$ $Constraint^{\Delta}_{I} = (\exists, \bot)$ $Constraint^{\Delta}_{\Sigma_2} = (\forall, \lvert InstanceSet^{\Delta}_{\Sigma_2}\rvert)$ $Constraint^{\Delta}_{O} = (\exists, \bot)$ $InstanceSet^{\Delta}_{\Sigma_1} = \{s \in \Sigma \mid s \text{ is a conversational source state}\}$ $InstanceSet^{\Delta}_{\Sigma_2} = \{s \in \Sigma \mid s \text{ is a conversational target state}\}$

Table 6.1: Example of mapping the prosaic description of a design pattern to a structured representation.

Definition of UI Design Patterns using a structured representation. The following section describes how the patterns were modelled using the structured representation. Details concerning the mapping can be found in Appendix C.

At first, the prosaic description of the pattern was analysed for the existence of specific statements concerning the pattern's characteristics. The described characteristics were then mapped to the layer(s) they refer to, and finally reformulated using constraints. Table 6.1 provides an example of a part of the pattern.

As part of the study, the various *InstanceSets* have been identified using heuristics. In a forward engineering approach, these sets could be populated by the designer (cf. Chapter 7).

6.3.4 How many Design Prototypes from professional software engineering projects contain implicit requirements? (RQ 2.3)

In order to answer this question, heuristics were developed for identifying potential instances of UI Design Patterns in the given archival data (cf. Appendix C.1.4). The potential instances were then matched against the design patterns with a software implementation of the approach in Section 5.4.2. The study focused on the identification of four UI Design Patterns. Out of the 49 prototypes, 22 contained the *Global Navigation* pattern, 18 contained the *Pyramid Pattern* and 1 contained the *Wizard Pattern*. The Sequence Pattern was even prevalent in almost all prototypes, which is due to overly eager identification heuristics as discussed below in Section 6.3.5.

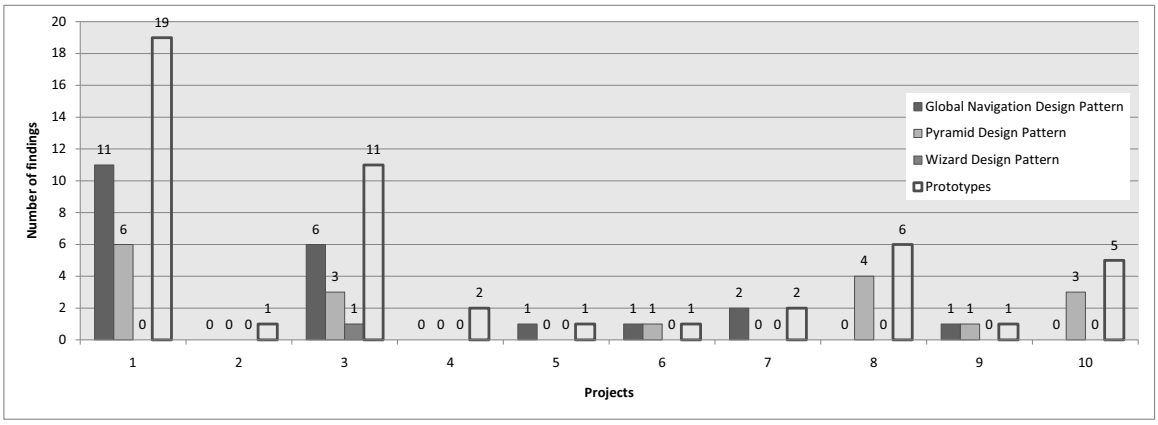

Figure 6.4: The number of UI Design Patterns that have been identified is evenly distributed among prototypes from almost all projects under examination (prototypes and findings aggregated by project).

6.3.5 Evaluation of Validity

Threats to *construct validity* are imposed by the fact that the study data was not originally created for the study and its research questions. It is therefore not clear a priori, whether the Design Prototypes were created for *a)* IDSs and *b)* with UI Design Patterns in mind. *Internal validity* is limited by the influence of the software tools used for the analysis. As there is no proof of correctness for the software tool, non-existing ones might still be identified (*false positives*). UI Design Patterns that might remain unidentified (*false negatives*) are not considered a threat as the study results indicate a lower bound on the occurrences of UI Design Patterns in Design Prototypes. Another threat comes from the definition of the UI Design Patterns themselves, as the software tool is only capable of identifying what has been modelled. *External validity* is not necessarily given as the Electronic Prototypes have been created using a specific design tool (Pidoco). Using other electronic prototyping tools (e.g. Balsamiq, Axure, Powerpoint, etc.) might alter the resulting artefacts and cause different results.

The threats to *construct validity and internal validity* have been addressed by asking usability experts from Pidoco GmbH for a review of the results. The experts were provided with random samples of the results and asked to assess whether the identified patterns were actually present in the prototypes and whether there were false positives. The results indicate that the random samples contained false positives for the sequence map pattern. After discussing this finding with the experts, it was concluded that the heuristics for identifying sequence map patterns were overly eager and the respective results have been discarded from the findings of this study. *External validity* is assumed to be given as the artefacts that are created with Pidoco and other prototyping tools can easily be mapped to the generic model for Electronic Prototypes as described in Chapter 3. It has been the basis for the definition of the structured representation that was used in the study. It is therefore assumed that the study could be repeated even based on other archival data from different prototyping tools with no significant changes in the qualitative results. The absolute number of

patterns being identified, however, strictly relies on the design projects themselves, i.e., other projects will yield other numbers of UI Design Patterns.

6.4 Conclusions

Summary of Conclusions

The results indicate:

1. Realistic Electronic Prototypes can be mapped to a structured representation for Design Prototypes.
2. Realistic Electronic Prototypes contain significant numbers of UI Design Patterns.
3. UI Design Patterns contained in realistic Electronic Prototypes can be mapped to a structured representation for UI Design Patterns

A structured representation for Design Prototypes is a possible way for an automated analysis of Design Prototypes. This is a prerequisite for a more continuous integration of prototyping activities into the design of IDSs and thus for a more continuous integration of users.

Although the study was conducted at a very low scale with only *four* different UI Design Patterns, the results were so extensive that it was only possible to validate samples of the results with experts. Therefore, it is necessary to provide tool support for identifying and tracing implicit requirements in Design Prototypes to avoid loss of requirements during the early stages of the design of IDSs.

Relation to Existing Evidence

The exploratory case study in Chapter 4 indicated that Design Prototypes are an unstructured representation of IDSs that provides suboptimal descriptiveness. The efforts necessary for integrating Design Prototypes in an iterative process are increased by the efforts necessary to document the information that is contained in these artefacts. The study presented in this chapter indicates that it is possible to translate electronic prototypes to a structured representation that can be used to identify implicit requirements, namely UI Design Patterns.

Impact/Implications

The study suggests that *professional projects working with UI Prototyping methods are confronted with*

- *either significant manual effort for translating the unstructured Design Prototypes into a more formal representation*
- *or risking the loss of requirements along the way.*

Limitations

The subjects were selected based on availability. It is therefore possible (and likely) that other projects will differ in their parameters, just like the selected subjects differed with each other. It can be assumed that the differences would have no significant impact on the results of this study, as the detailed data for this study suggests that the identified effects are the same for all subjects irrespective of their size. Another potential is the selection of the design patterns used in this study. It is not assumed that the effects would be the same if other design patterns would have been selected. Design patterns that focus merely on visual requirements, for example, might be harder to model or even impossible to model with this approach. Nevertheless, even if the results of this study would not be applicable to a large number of other design patterns, the approach is generic enough to be applied to other types of structural or behavioural requirements as well.

If a design is going to be iterated - as it will - then it is very important that each cycle through the design process not have to start again from scratch.

Harald Thimbleby

7 A Workflow for Continuous Integration of Users into the Design of IDSs

Chapter 6 indicated that Design Prototypes in realistic software engineering projects actually contain implicit requirements such as UI Design Patterns. This itself does not pose a problem, though, unless the implicit requirements are explicitly documented and traced. A first step towards supporting the explicit documentation was the identification of a reference architecture (Chapter 5), which allows to map the informal artefacts to a semi-formal model.

The next step will be to examine how this approach can be integrated into model-based RE. In 2010 a empirical study[1] was conducted among enterprises, that develop software for software-intensive, embedded systems, to identify their needs for a model-based approach to RE. The results indicate that a major challenge remains the methodological support for developing requirements across layers of abstraction, and to make sure that they are consistent, testable and complete [79].

Recent work indicates that the evaluation of requirements needs to be done continuously throughout the software development process [18]. With respect to user-related requirements, recent findings additionally suggest, that the sample size (in terms of test users and test cases) required to eliminate most user-related issues is much higher [81] than previously expected [86].

This chapter therefore introduces a workflow for the continuous and seamless integration of Usability Activities (and thus of users) into the design of IDSs, which supports a better documentation and traceability of requirements in Design Prototypes across design iterations. It is the basis for a prototypical tool support (Chapter 8) and a case study in a realistic software engineering project (Chapter 9).

7.1 Open Issues

Taking a closer look at the overall goals for this work as described in Chapter 1, open issues can be identified with respect to the *continuity* and *seamlessness* of the

[1] The study has been conducted as part of project SPES 2020 (http://spes2020.informatik.tu-muenchen.de/) between May 2009 and January 2010.

integration of users into the design of IDSs.

Limiting factors for continuity.

1. **Milestone-orientation.** Evaluation of requirements with users is usually conducted at milestones only [88]. With respect to Usability Activities users are primarily integrated for usability tests. Users hold sticky information (cf. Section 3.3.2) concerning requirements that can hardly be elicited by a passive involvement or milestone-oriented interviewing.
2. **Manual effort.** There is no tool support for (semi-)automatic validation of user-related requirements. This lack of semi-automatic mechanisms prohibits the continuous evaluation of quality criteria that cannot be evaluated automatically. Some of the reasons such as the unstructured representation of Design Prototypes and implicit requirements have been discussed in Chapters 4 and 6.

Limiting factors for seamlessness.

1. **Missing links between artefacts.** It is unclear what the formal relationship between different types of prototypes is. Furthermore, there is no mapping between Design Prototypes that are used for Usability Activities and more formal representations of IDSs. Dix describes this as the "formality gap", "the conflict between the need for rapid turnaround and fast prototypes, and formal refinement". [22, p. 301]
2. **Missing links between tools.** There is no connection between typical tools for UI Prototyping (e.g. paper prototyping tools, electronic prototyping tools) and state-of-the-art RE tool chains. This holds both for Usability Activities for requirements elicitation and evaluation.
3. **Missing links between activities.** The integration of Usability Activities into software engineering is an open issue. Advances have been made, e.g. by formalizing usability requirements and integrating their verification into the software engineering process [90]. However, the elicitation and validation of user-related requirements in a way that allows for applying more formal software engineering methods is still missing.

7.2 Requirements

The workflow that is introduced below addresses the *issues* and *considerations* described above by abiding the following specific requirements:

- The stages of the workflow need to be integrated with *established RE activities*.
- *Usability Activities* such as UI Prototyping need to be integrated and need to remain *fast and disposable*.
- Users need to be integrated both for *elicitation* and *validation* of requirements.
- Different types of Design Prototypes need to be supported seamlessly across RE iterations.
- Requirements in Design Prototypes need to be *identified, traced and evaluated* in a *continuous* manor.

- It needs to be clarified how *work artefacts* are related and how they can be integrated into other software engineering activities *seamlessly*.
- *Manual effort* for identifying requirements needs to be reduced.
- Tools for Usability Activities and RE need to be integrated.

7.3 Phases

The following section introduces the phases of the suggested workflow. It is based on the general workflow for requirements engineering as discussed in Section 3.1.

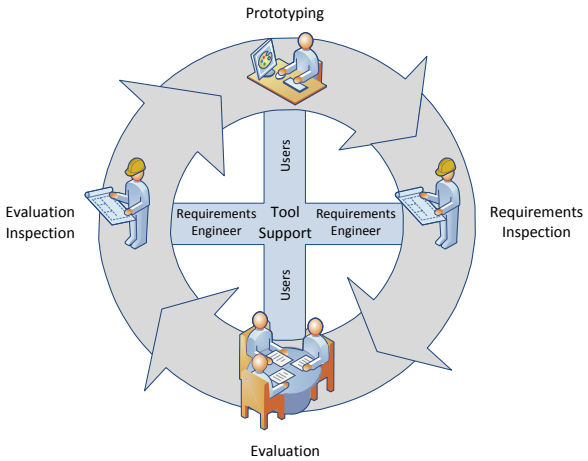

Figure 7.1: Overview of the suggested workflow for continuous and seamless integration of users.

7.3.1 Overview

The workflow is defined by an iterative process, which consists of four phases: *(1) Prototyping, (2) Requirements Inspection, (3) Evaluation, (4) Evaluation Inspection* (Figure 7.1). These phases incorporate those Usability Activities, which are best suited for integrating users according to DATech [20]: The *Prototyping* phase allows for the *discussion of the application concept* and a *collaboration on prototypes*. The *Evaluation* phase allows for the *validation of requirements* and *User Testing*. With respect to RE, it can be stated that the *Prototyping* phase is part of the *Requirements Elicitation* , the *Requirements Inspection* phase is part of the *Requirements Analysis*, the *Requirements Evaluation* phase is part of the *Requirements Validation*, and the *Evaluation Inspection* is part of the *Requirements Management* in RE. It should be noted, however, that the suggested workflow is intended for *extending* existing RE activities. It is *not* (and could not be) intended to replace the mentioned RE activities. Table 7.1 summarizes these relationships.

The following sections describe in detail the four phases of the suggested workflow for the integration of users into the design of IDSs.

Workflow Phase	...incorporates Usability Activities [20]	...is part of RE activity [80]
Prototyping	UI Prototyping	Elicitation
Requirements Inspection		Analysis
Evaluation	User Testing	Validation
Evaluation Inspection		Management

Table 7.1: Relationship between the suggested Workflow, Usability Activities and RE.

7.3.2 Prototyping

The elicitation of requirements usually consists of two stages: requirements discovery and requirements classification. The *prototyping phase* contributes to both stages (Figure 7.2).

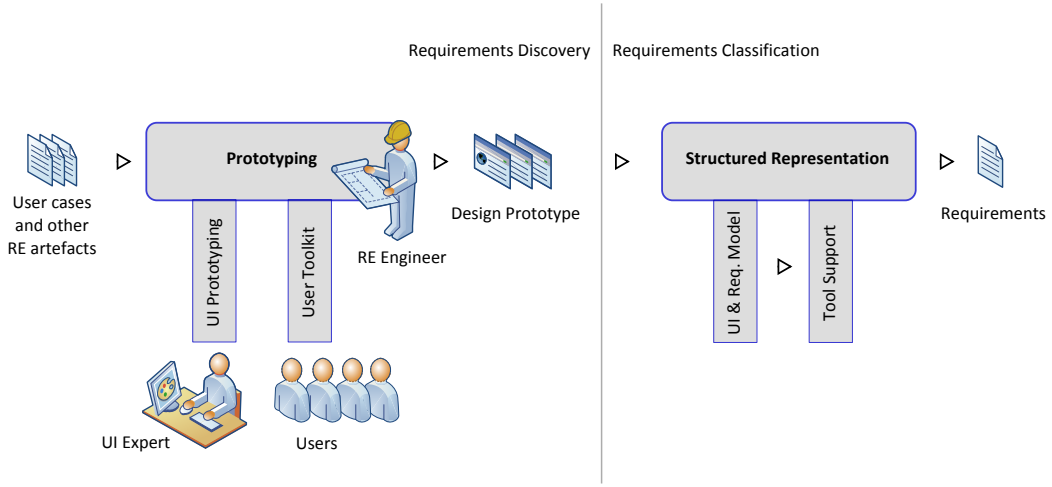

Figure 7.2: The prototyping phase includes activities for UI Prototyping and the structured interpretation of the resulting Design Prototypes.

Prototyping. The prototyping stage comprises the regular UI Prototyping techniques described in Section 3.3.1. UI Prototyping can be used to elicit user-dependent requirements concerning use cases, user tasks and the UI. It can be based on requirements that have been gathered by other existing methods for RE discovery, e.g. interviews, or previous RE iterations. A requirements engineer is responsible for conducting the prototyping sessions. UI Prototyping typically involves UI experts and users such as end users, the customer, internal and external test users. As part of the sessions Design Prototypes are created which contain storyboards of individual user tasks and mockups of individual screens (or the respective artefacts of non-graphical modalities). If users should be able to contribute to the creation of the prototypes directly, a User Toolkit is needed (cf. Section 3.3.2). At this stage (as opposed to later stages in the software engineering process), the requirements concerning the UI are documented by the Design Prototype itself and need to be extracted.

Structured Representation. The resulting Design Prototypes are then mapped to a structured representation (as introduced in Chapter 5) in the structured representation stage. This includes RE classification, i.e. organization and grouping of user-related requirements. This work focuses on two types of requirements: *a)* user tasks and architectural requirements, which are reflected by the structured representation of the Design Prototype and *b)* UI Design Patterns, which are reflected by their structured representation. Although this stage can be conducted manually, the inherent complexity of Design Prototypes (cf. Chapter 4 and 6) suggests that it should be supported by tools that help with *a)* the conversion of the Design Prototypes from their original format (e.g. paper prototypes, electronic prototypes) to a digitized version and from there to a structured representation and *b)* with the identification of implicit requirements such as UI Design Patterns.

7.3.3 Requirements Inspection

The *requirements inspection phase* contributes to RE Prioritization/Negotiation and RE Specification (Figure 7.3).

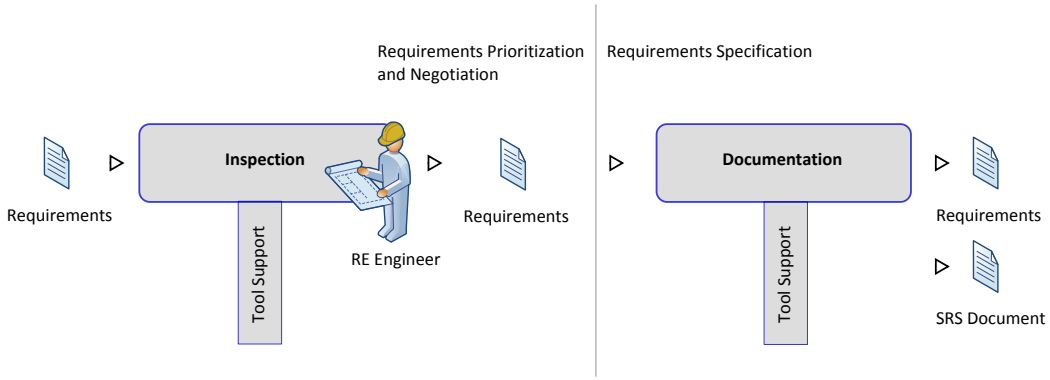

Figure 7.3: The requirements engineer reviews the elicited requirements during the requirements inspection phase.

Inspection. In the review stage, a requirements engineer inspects and manipulates the requirements from the elicitation phase. This is necessary as the artefacts that have been created in the prototyping phase are inherently ambiguous in nature which means that the results of the analysis of these artefacts need to be inspected for their correct interpretation. In order for this to be feasible, the requirements need to be presented in a human readable way. This can be achieved by mapping the requirements to the reference architecture presented in Chapter 5 and visualizing the mapping layer-wise (cf. Figure 7.4). The requirements engineer should have the tools necessary to adjust or add requirements, e.g. by defining the *InstanceSets* of UI Design Patterns.

Documentation. The inspected requirements are then documented, both in terms of a structured representation of the requirements and in a human readable way in the form of generating segments for a software requirements specification document. The

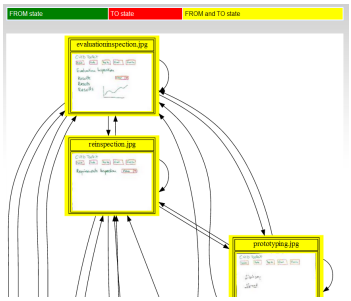

Figure 7.4: Example of a visualization of *InstanceSets* of a design pattern on the conversational layer.

segments that can be contributed by this workflow are: user interface requirements represented by the structured representation of Design Prototypes and UI Design Patterns, product functions represented by the storyboard view of a Design Prototype and the test cases (defined in the Evaluation phase, see below).

7.3.4 Evaluation

In the *evaluation phase* the requirements that have been recorded in previous phases are *validated* with users. Validation is important, as requirements might have been left out during the elicitation, specified incorrectly and, according to Sommerville [80] users may have conflicting and even changing requirements. As far as possible, checking of requirements can be conducted for semi-formal specifications of requirements against semi-formal specifications of Design Prototypes. The results are then mapped to the respective requirements and the related elements of the reference architecture (Figure 7.5).

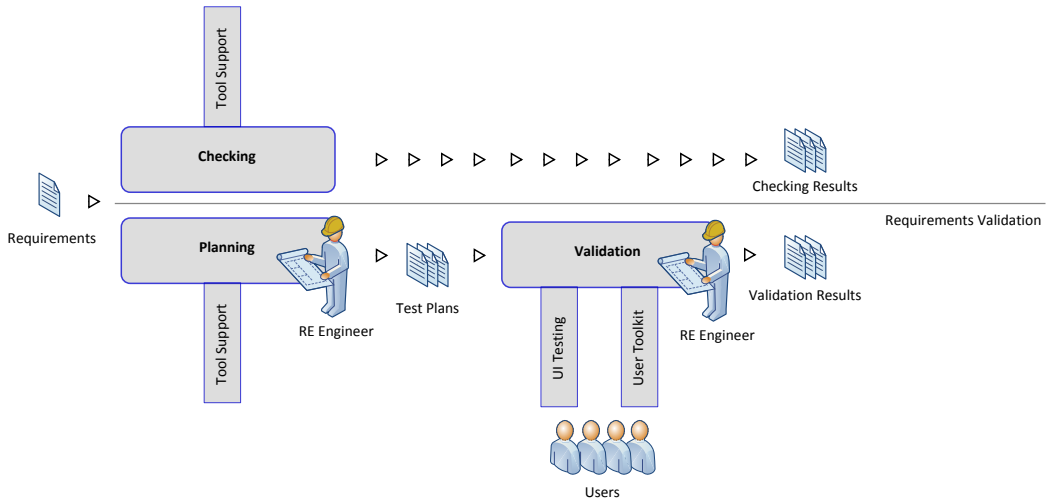

Figure 7.5: Checking a design prototype for conformance with its requirements and validating requirements is part of the evaluation phase.

Planning. During the planning stage, the requirements engineer creates test plans that contain test cases for all requirements that are to be validated with users. A test plan refers to a specific Design Prototype and is described by a *goal*, a *description*, a conversational start state and a conversational target state. The latter are defined by selecting mockup screens from the Design Prototype.

Validation. The validation is then conducted by the requirements engineer together with selected users. Two possible methods for performing the validation, which have been described in this work, are User Testing and the use of User Toolkits for remote User Testing. The validation for a specific use case is led in by asking the user to perform the actions necessary to reach the given *goal* with the help of the Design Prototype starting from the conversational start state. Explicit user feedback and observations by the requirements engineer are documented in relation to the conversational state they occur in. The results from the individual validation sessions are documented by grouping test plans with their test cases and the results from the related validation sessions.

Checking. In a model-based approach to UI Design automatic, model-based verification of requirements such as usability requirements is possible as described by Winter [90]. The approach in this work provides an intermediate step towards the definition of formal models for IDS. The structured representations developed in this work can be used to perform semi-formal checks of Design Prototypes with respect to structured representations of user-independent requirements such as UI Design Patterns. Although the description techniques developed in this work are mostly semi-formal [13, p. 9], the modality-independent layers of the reference architecture and UI Design Patterns for these layers can be specified semi-formally and be used for checking Design Prototypes for UI Design Patterns as described by the *match* function (see Function 5.2). The checking is performed automatically and the results are documented by grouping prototypes together with the checked requirements and the identified matches.

7.3.5 Evaluation Inspection

In the *evaluation inspection* phase the evaluation results and their interpretations are then reviewed by a requirements engineer in order to define *change proposals* for the next prototyping phase (Figure 7.6).

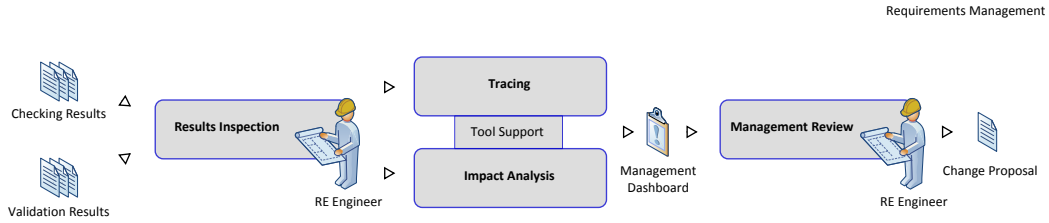

Figure 7.6: The results of the evaluation and their impact is assessed in the evaluation inspection phase.

Results Inspection. During the results inspection stage, the requirements engineer reviews the results from the validation and checking stages. The checking results indicate which requirements could be checked and to which extent. The validation results provide an overview of feedback from User Testing.

Tracing. During the tracing stage, checked requirements are examined for their management status, i.e. requirements are grouped into *new*, *removed*, *changed* and *unchanged* requirements. This is especially helpful for requirements that stem from previous RE iterations, potentially involving a change of prototype format from paper to electronic in the meantime. The management status is then part of the Management Dashboard, which provides an overview of the current tracing status for all requirements (cf. Figure 7.7).

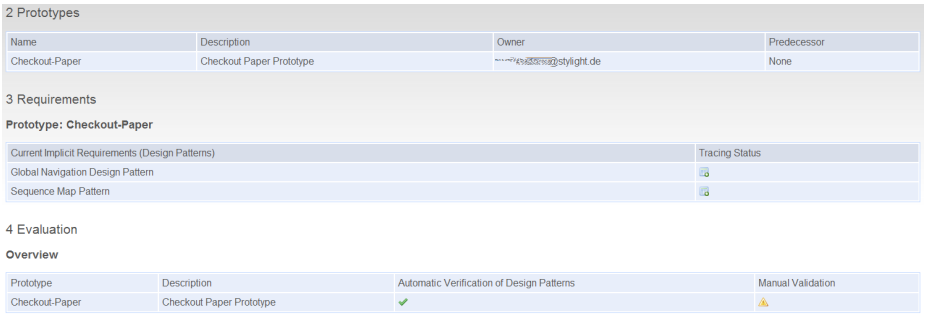

Figure 7.7: The management dashboard provides an overview of the current evaluation and tracing status of the elicited requirements.

Impact Analysis. The impact analysis correlates requirements and validation results. This is helpful in order to understand the mutual effects changed requirements may have. The results of the analysis are part of the Management Dashboard and are visualized to highlight which parts of a Design Prototype are affected by validation and/or checking results (cf. Figure 7.8).

Management Review. To the end of an RE iteration, the requirements engineer uses the Management Dashboard to inspect the tracing status of requirements and the impact analysis of validation results. Based on the insights gathered, a change proposal is formulated in written form, which describes suggested changes for the next RE iteration. Controlling the quality of software artefacts based on predefined quality models makes necessary *continued manual reviews by quality engineers* [19]. This insight can be transferred to controlling the quality of Design Prototypes with respect to previously defined requirements. Therefore, the Management Dashboard should be available to the requirements engineer continuously, in order to be able to assess the quality of a Design Prototype at all times.

7.4 Summary

Table 7.2 provides an overview of the artefacts and their representation for each stage of the workflow. The suggested workflow addresses the requirements identified

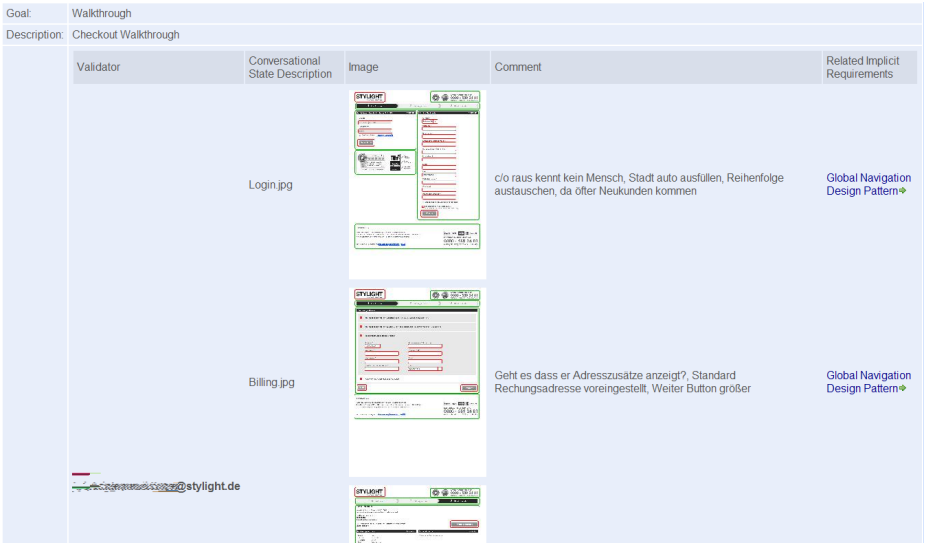

Figure 7.8: Visualization of validation results and their relation to the conversation layer and UI Design Patterns.

in Section 7.2 as summarized by Table 7.3. The workflow describes in detail how the structured representation for Design Prototypes and UI Design Patterns can be used as a basis for a more continuous and seamless integration of users into the design of IDSs. In order to examine whether the workflow actually provides positive impact on realistic software engineering projects, its application is documented as part of a case study in Chapter 9. Chapter 8 describes a prototypical implementation of the respective tool support.

Stage	Resulting Artefact	Representation
UI Prototyping	Design Prototype	Informal (Paper, Wireframes, ...)
Structured Representation	Requirements	Structured (Chapter 5)
Inspection		
Documentation	Requirements, SRS	Structured + HTML Document
Checking	Matches	Structured (Chapter 5)
Planning	Test Plans	Form-based
Validation	User Feedback	Form-based
Results Inspection	–	–
Tracing	Management Status	Management Dashboard
Impact Analysis	Related Requirements	Management Dashboard
Management Review	Change Proposal	HTML Document

Table 7.2: Overview of stages and artefacts of the suggested approach.

Requirement	Addressed by
Integration of Usability Activities	UI Prototyping stage, validation stage
User integration in elicitation and validation	User Toolkit for UI Prototyping and validation stage, User Testing for validation stage
Seamless integration of different types of Design Prototypes	Structured representation and tracing stage
Continuous requirements tracing	Tracing stage and Management dashboard
Relation of artefacts	Explicitly defined by the artefact-based workflow
Reduced manual effort	Tool support and structured representation
Tool integration	Tool support (Chapter 8)

Table 7.3: Review of the workflow with respect to its requirements.

Transform complexity into products that are understandable, as such things are judged to be simple.

Don Norman

8 Tool Support CUID

Chapter 7 discussed a workflow for continuous and seamless integration of users into the design of IDSs. This chapter introduces CUID — a *c*ontinuous *u*ser *i*nterface *d*esign toolkit, which supports the complete workflow. This also includes User Toolkit functionality, which allows non-expert users to take part in the activities while preserving a structured representation of the requirements that are elicited. Figure 8.1 provides an overview of the interactions that users and experts are able to perform with the toolkit and the conceptual inner workings that provide for the seamlessness between work artefacts and requirements artefacts. CUID was applied to a practical software engineering project in a case study (cf. Chapter 9).

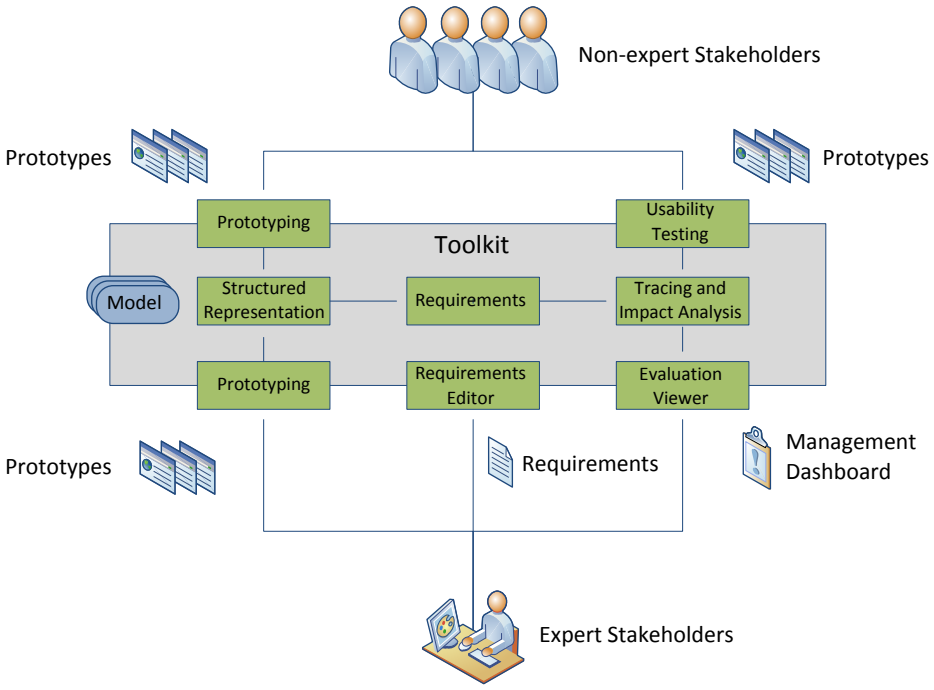

Figure 8.1: Schematic overview of Collaborative UI Design.

8.1 Requirements

CUID is designed based on requirements that can be categorized into three groups: *workflow requirements*, *prototyping tool requirements* and *user toolkit requirements.* Each requirement is assigned an identifier that will be used in the remainder of this chapter. The identifier contains a letter (F: Feature, A: Architecture, I: Implementation) and a number. The notation is used to map features, architectural properties and implementation details to the requirements that they relate to.

8.1.1 Workflow Requirements

This group of requirements is deduced by analysing the workflow introduced in Chapter 7 with respect to its design and the role tool support is supposed to fulfil. In addition this group also considers the requirements that arise from the fact that the suggested workflow needs to be validated in realistic software projects.

Complete workflow support (F1). The tool should support the complete workflow in order to guide the requirements engineer through the suggested process. As the workflow is based on an iterative process model with the goals of *seamlessness* and *continuity*, a partial coverage of the workflow would not allow to use CUID for a confirmatory or comparative case study.

Integration with existing tools (A1). The workflow is designed to be integrated with existing RE processes and tool chains. This carries forward to the use of CUID for implementing specific stages of the workflow.

Minimal setup (I1). In order to allow for a timely deployment in a case study environment, the tool should be minimal invasive within its environment and therefore quick to set up.

Parallel design (F2). CUID should be capable of supporting parallel design, i.e. it should be possible to create multiple prototypes within a single iteration and to define multiple prototypes as being refinements of single prototypes of previous iterations.

User integration (F3). The tool should provide *User Toolkit functionality* in order to allow the integration of users into the process, specifically for elicitation and validation of requirements.

Collaborative UI Design (F4). Besides integrating users, it is also of interest to integrate other stakeholders such as colleagues and software engineers into the project. The tool should support sharing project data and attributing contributions to individual team members.

Remote access (I2). Providing remote access to CUID is a technical prerequisite for *continuous involvement* of users and other stakeholders.

Extensibility (A2). CUID is currently a functional prototype with horizontal coverage of the complete workflow and vertical implementation of specific functionalities. At the moment it is limited e.g. to specific prototyping methods and a limited number

of UI Design Patterns. The tool should allow for extension towards other prototyping methods and other patterns.

8.1.2 UI Prototyping Requirements

Cuid is in part a UI Prototyping tool. Therefore various requirements, that have been identified for this category of tools [82] also apply here. Some of these requirements are already contained in the *workflow requirements* and are therefore not repeated here, namely: *lifecycle support* (in the sense of integrating the prototyping tool with the existing tool chain), *team design* and *version control* (which is a prerequisite for supporting parallel design).

Ease of Use (F5). One of the differences between Design Prototypes and functional prototypes is the effort necessary to develop them. Design Prototypes are used in the first iterations of the RE process and the tools used for creating them should be effortless to use.

Fast Turn-Around (F6). UI Prototyping is a method for *fast* creation of *disposable* Design Prototypes. This is due to the fact that prototyping is targeted at examining many different design options within a given design space.

Extensive Control over Prototype Features (F7). The expressiveness of Design Prototypes should not be limited by the prototyping tool. Chapter 4 suggests that paper prototyping has advantages in terms of its expressiveness towards electronic prototyping.

Data Collection Capabilities (I3). The prototyping tool should be capable of collecting data, which is generated e.g. by User Testing.

Executable Prototypes (F8). The Design Prototypes themselves should be executable in order to allow users to experience them. This is espescially important for validation activities.

8.1.3 User Toolkit Requirements

As already discussed in Chapter 7 Cuid provides User Toolkit functionality for the elicitation and validation of requirements. The requirements for effective User Toolkits by v. Hippel and Katz [39] will be considered for the design of these functionalities.

Learning by Doing via Trial-and-Error (F9). Users, who are motivated to contribute substantially to the design of an IDS such as internal test users, co-workers or the customer, should be provided with functionalities that allow them to create, experience and evaluate their own Design Prototypes.

An Appropriate "Solution Space" (F10). The possibilities provided to the user should not exceed the possible final product solutions in terms of functionalities and

other features. The User Toolkit should only offer tools that allow for the design of realistic solutions.

User-Friendliness (F11). The User Toolkit should be appropriate for its user, i.e. it should refrain from requiring expert knowledge concerning software engineering and instead allow the user to primarily rely on his already acquired skills.

Module Libraries (F12). Design Prototypes often share common features that can be offered to the user as modules for integration in Design Prototypes, e.g. stencils for GUI elements or auditory icons for the design of VUIs.

Translating User Designs for Production (A3). "the *language* of a toolkit for user innovation must be convertible without error into the *language* of the intended production system at the conclusion of the user design work." [39] This requirement is a requirement that is already at the core of this work: the seamless integration of informal Design Prototypes with more formal structured representations.

8.2 Feature Overview

The following provides an overview of the features that have been implemented in the CUID prototype. They were designed to address all feature related requirements.

8.2.1 Project Configuration

CUID supports the requirements engineer with basic project configuration tasks (cf. Figure 8.2). This is necessary as a basis for all subsequent workflow support features.

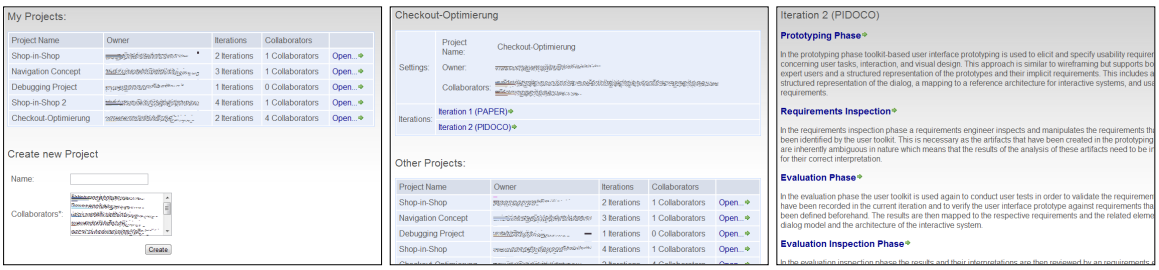

Figure 8.2: CUID supports basic project configuration with collaborators and RE iterations.

Projects (F4). Each *project* can be configured by its *owner* by adding *collaborators* to the project. This provides collaborators with access to all features of CUID for the given project.

Iterations (F2). Each project consists of a number of *iterations*. Iterations are categorized based on their respective prototyping method. CUID supports iterations based on paper prototyping and electronic prototyping with Pidoco. Each iteration

consists of the four workflow phases defined in Chapter 7 and supports the simultaneous creation of multiple Design Prototypes.

8.2.2 Workflow Support

CUID provides support for the complete workflow, for the requirements engineer, collaborators and users. The following describes the functionalities in detail.

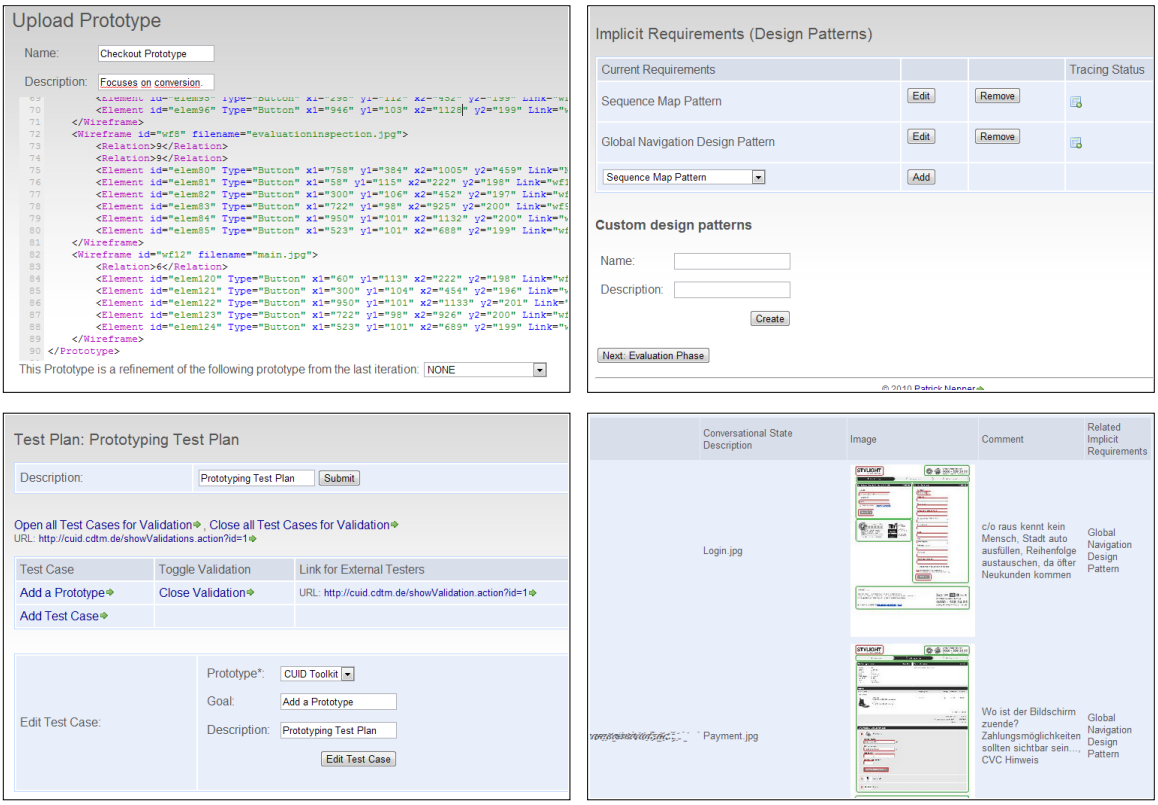

Figure 8.3: Top left: Importing a digitized Design Prototype to CUID. Top right: Overview of implicit requirements. Bottom left: Creating a test plan for validators. Bottom right: Inspecting validation results.

Prototyping (F1, F5-8). One of the requirements for CUID was to support existing prototyping methods and tools. The prototypical implementation of CUID supports both paper prototyping and electronic prototyping. Design Prototypes that have been created with a prototyping tool can be imported into CUID by uploading an XML representation of the prototype and the respective mockups and storyboard images. While an XML representation is trivial for electronic prototypes, paper prototypes first need to be digitized and converted to an XML representation.

A tool for converting paper prototypes to an XML representation (PAPER2XML) has been developed at CDTM [77] and has been integrated into CUID (see Figure 8.4). PAPER2XML can be used to semi-automatically detect GUI elements (e.g. buttons,

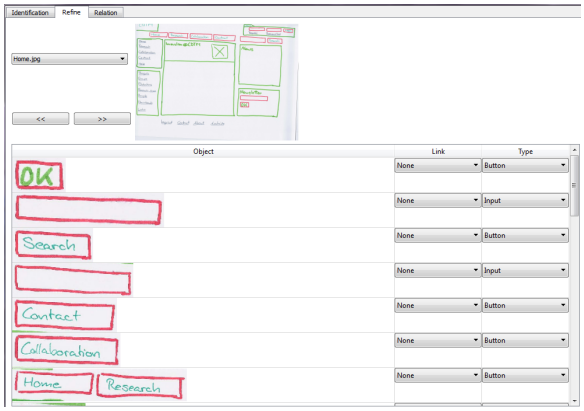

Figure 8.4: PAPER2XML [77] supports semi-automatic conversion of Paper Prototypes to XML.

checkboxes, links, etc.) in mockups and to define a click model based on the identified GUI elements. This is achived by applying computer vision algorithms to the paper-based Design Prototype. In order to allow for a high recognition rate, Design Prototypes need to be colour-coded to distinguish GUI elements from other visual artefacts.

Once a Design Prototype is uploaded it is automatically converted into a structured representation and heuristics are applied in order to identify UI Design Patterns. The requirements engineer can indicate whether the Design Prototype is a refinement of a Design Prototype from one of the previous iterations. CUID contains versioning functionality, i.e. revised Design Prototypes are not removed but remain accessible in the prototyping view of the respective iteration.

Requirements Inspection (F1). CUID provides an overview of currently traced UI Design Patterns for each prototype. It is possible to add UI Design Patterns to a given prototype, to edit existing patterns and to remove patterns. Each pattern is annotated with its current tracing status. It is theoretically possible to define custom UI Design Patterns, but the respective prototypical tool support does not provide a user friendly user interface.

Evaluation (F1). The evaluation phase is supported by offering checks for UI Design Patterns and support for the definition of test plans for validation. Checks are performed on patterns that have been specified in the prototyping phase. Test plans can be created as described in Section 7.3.4. CUID automatically generates URLs that provide access to a User Toolkit environment for conducting user tests (see below).

Evaluation Inspection (F1). CUID provides a management dashboard that provides an overview of the status of individual prototypes and their requirements with respect to checks, impact and tracing. The tracing view shows which requirements have been new, edited, removed or remained unchanged since the last iteration. The check view shows to which degree specific UI Design Patterns could be matched with

the structured representation of a given prototype. The impact view shows validation feedback together with its impact on the prototype and its UI Design Patterns.

8.2.3 User Toolkit Functionalities

In order to allow users to take part in the process, CUID includes two features for integrating users: User Toolkit-based prototyping and User Toolkit-based remote User Testing.

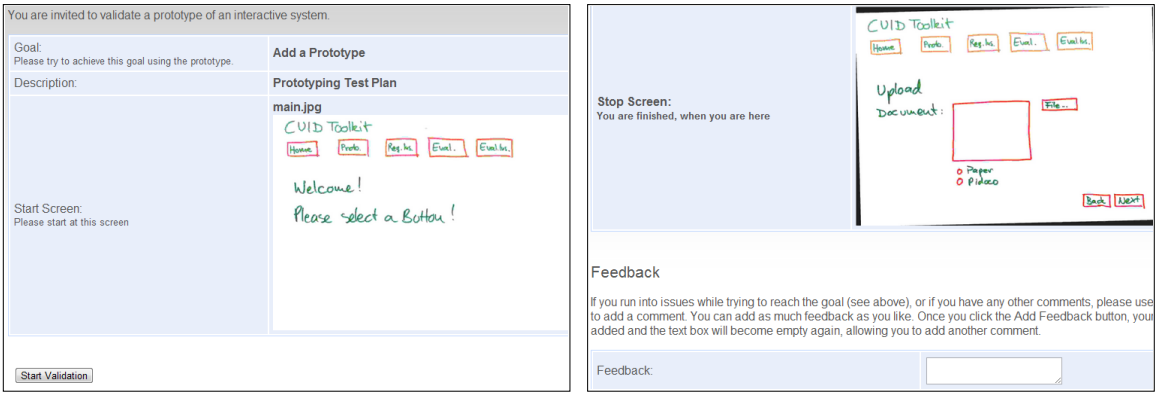

Figure 8.5: CUID integrates users through web-based validation of Design Prototypes.

Elicitation (F3, F9-12). There are existing tools that support users in taking part in UI Prototyping. Users can be involved in Paper Prototyping, which provides them with appropriate tools and the opportunity to experience Design Prototypes. Electronic prototyping tools with remote collaboration features such as Pidoco even allow to collocate the elicitation activities with the user in his usual environment. These tools provide modules for the creation of Design Prototypes such as specific stencils and prepared graphics. As both tools are already existing, CUID provides the necessary features to incorporate these tools into the workflow. The non-trivial integration of paper prototyping is achieved by providing a client software for converting paper prototypes to a digitized representation [77].

Validation (F3, F9, F11). CUID provides a web interface for users which allows them to take part in validation activities. The user interface is minimalistic and hides all complexities of the workflow and its data model. Users are provided with an overview of test cases, in which they can take part. Each test case is described by its goal and description. The user is asked to try to reach the goal starting from a given start state, which is represented by the related mockup. The user can provide feedback for individual mockups in text form. The user test itself is performed either onsite using User Testing, or remotely using remote User Testing (which is e.g. supported by Pidoco). CUID is used for documentation only.

8.3 Architecture and Implementation

The architecture has been influenced by the features described above and the architectural requirements.

8.3.1 Architecture

CUID is designed using a classical three-tier architecture.

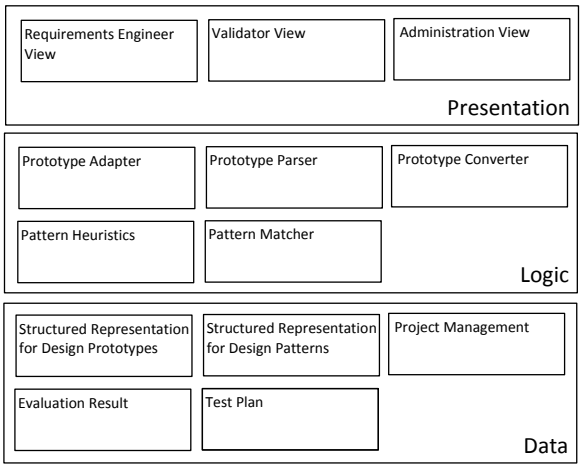

Figure 8.6: CUID can be extended by adding *prototype adapters, prototype parsers, design patterns* and *pattern heuristics*.

Presentation Layer. The *presentation layer* contains the views for the requirements engineer (and collaborators), validators and administrators. They contain the visible elements of the features described above.

Logic Layer. The *logic layer* contains the business logic of CUID. The translation of user designs **(A3)** is handled by the prototype parser and the prototype converter components: *prototyping adapters* are defined that generate an XML representation from a given Design Prototype. The *prototype parser* is responsible for parsing the XML representation of Design Prototypes and mapping it to a general data model for informal Design Prototypes. This mechanism allows for the integration with existing prototyping tools **(A1)**. The *prototype converter* generates a structured representation of prototypes from this informal representation. The UI Design Patterns heuristics component contains the heuristics that are used to identify *InstanceSets* of UI Design Patterns in Design Prototypes. The UI Design Patterns matcher checks whether *InstanceSets* match a given pattern. The heuristics component is defined by a generic interface which allows for easy extensibility for additional heuristics **(A2)**. The workflow controllers provide views with functionality.

Data Layer. The data layer contains the data models for the structured representation of Design Prototypes, UI Design Patterns, evaluation results, test plans, and general project management. The data layer makes use of interfaces for describing

UI Design Patterns in order to allow for extensibility with respect to design patterns **(A2)**.

8.3.2 Implementation

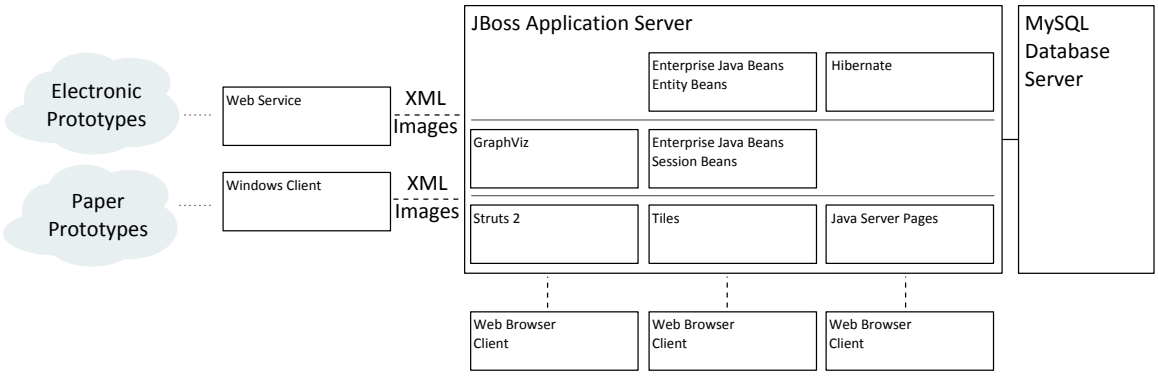

Figure 8.7: The prototypical implementation that was used for a comparative study (cf. Chapter 9) integrates Paper2Xml and Pidoco for UI Prototyping).

The prototypical implementation of Cuid combines existing tools such as paper prototyping, a windows client software for converting paper prototypes to XML representations, and Pidoco with a custom implementation of the features described above. Minimal setup **(I1)** was achieved by implementing Cuid as a web application. It was successfully applied in a case study (Chapter 9) without the need for setup work at the site of the study. The implementation as a web application also allows users to take part in the workflow from the office, from home or from the target context of individual use cases **(I2)**. Any prototyping tool that is capable of exporting an XML representation of Design Prototypes can be attached to Cuid. As part of the prototypical implementation this has been done for prototypes created on paper and with Pidoco. The windows client for the conversion of paper prototypes to XML [77] provides features for the automatic identification of GUI elements that are contained in mockups (see Figure 8.4). The MySQL database contains tables that track usage statistics for the web application, which is necessary to infer conclusions concerning the efficiency and effectiveness of the tool support **(I3)**. The technologies used in the implementation are standard technologies for Java-based software development.

8.4 Summary

Cuid is a prototypical tool-support for the workflow presented in Chapter 7. It provides all features necessary to collaboratively create Design Prototypes and validate them with users. A major advantage of using the tool support for an RE project is the ability to trace requirements across multiple RE iterations and across different types of Design Prototypes, e.g. from paper prototype to electronic prototype. Although Cuid is currently limited to a small number of UI Prototyping methods and

UI Design Patterns, its architecture allows for an extension towards other methods and patterns. It has been designed with the goal to apply the workflow for seamless and continuous integration of users in a case study in a professional environment. The results of the study are documented in Chapter 9.

The tool standardizes the workflow for specification and user tests, which makes it more effective and simpler

study participant

9 Study: Validation of Continuous, Seamless Integration in a Professional Environment

This study validates the workflow and tool-support Cuid for continuous and seamless integration of users as introduced in Chapters 7 and 8 by applying it to a professional software engineering project. The study was conducted in late 2010 together with Pidoco GmbH, Berlin and Stylight GmbH, Munich. The results indicate that the suggested approach improved the identification of implicit requirements in informal Design Prototypes, the frequency of interactions with users and the requirements management in the study environment. As is the case with the other case studies in this work, the following is structured according to Runeson and Höst 2009 [70].

9.1 Introduction

In order to understand the effect of applying the previously described workflow and tool-support Cuid to a professional project, this study is designed as a comparative, improving case study facilitated by two projects at the same company: one project which applies the suggested approach and one project which leaves the given RE process unchanged. This allows for a direct comparison between the status quo and the suggested approach. The study is based on first-degree (interview) and second-degree methods (usage data collection and work artefacts). It also includes aspects of action research as it introduces a changed workflow and additional tool support at the partner company. The purpose of this study, however, is observational, i.e., this study documents the *effects* of the induced change, not the change process itself. It is therefore categorized as a case study and follows the respective methodology [70, p.134].

Problem Statement

The design of IDSs relies on a continuous and seamless integration of users into the process. The seamlessness is limited by the the formality gap between unstructured design artefacts (Design Prototypes) and formalized requirements. The continuity of the integration is limited by the effort necessary to overcome the formality gap and the absence of adequate tool support for user integration. Chapters 7 and 8 introduced a workflow and toolkit for improving this situation.

Research Objective

This study examines the impact of a workflow and tool support for continuous integration of users into the design of IDSs, specifically by assessing the integration of the suggested workflow and tool support into an existing engineering process and by examining the impact on requirements elicitation and user integration. The results of this study validate the contribution of this work and point to future work in this field.

Study Partners

The study was conducted at Stylight GmbH, Munich, and was accompanied by a diploma thesis by one of the managing directors. As in the previous case studies, Pidoco GmbH contributed its tools for electronic prototyping.

Pidoco GmbH. In this case study, Pidoco provided its prototyping tool and a *prototyping adaptor* as described in Section 8.3.1, which was used to convert Design Prototypes created with Pidoco to an XML representation for use with Cuid. Information on Pidoco's service and the company can be found in Sections 4.1 and 6.1.

Stylight GmbH. Stylight is a Munich-based e-commerce start-up company that was founded by alumni of CDTM in 2008 and which received Series A funding from Holtzbrinck Ventures (HBV) and Series B funding from HBV and Tengelmann E-Commerce. Stylight offers e-commerce solutions for online fashion stores, such as a marketplace with a unified shopping cart for various fashion shops and shopping widgets that can be embedded into existing web sites. The user can browse through more than 250.000 fashion items and buy more than 40.000 of these directly at Stylight. At the time of the survey, Stylight was run by a team of four founders and 25 (FTE) employees. Stylight supported the case study with the necessary human resources, by applying the suggested workflow and tool support and by sharing study data.

9.2 Case Study Design

Research Questions

This case study examines the third top level research question:

> **RQ 3** How can Design Prototypes be used more efficiently for eliciting and validating requirements?

This research question is subsequently split up into three aspects: integration of the suggested approach with existing prototyping chains, amount of requirements being traced, and frequency of user integration.

RQ 3.1 *How can the suggested approach be integrated with existing prototyping tools?*

The study examines whether the translation of paper prototypes and electronic prototypes to a structured representation can be achieved in a professional software development project.

RQ 3.2 *What is the impact of an improved workflow and tool-support on the amount of requirements being documented and traced?*

Using the suggested toolkit, Design Prototypes can be translated into a structured representation that makes implicit requirements explicit. This question addresses whether the amount of requirements being identified, documented and traced actually changes once the toolkit is used in a realistic setting.

RQ 3.3 *How does the frequency of integrating users into the design of an IDS change with an improved workflow?*

The workflow for continuous and seamless integration of users defines activities and artefacts that are targeted at a better integration of users and related activities into the RE process. This study examines the impact of the workflow in a realistic setting.

Case and Subjects Selection

The case was selected with the following goals in mind:

- prior experience with integrating users and prototyping methods to support external validity
- high frequency of design projects to allow for flexible study design and a selection of projects that supports construct and internal validity
- efficient access to study data to allow for in-depth analysis and a level of documentation that supports reliability

The study partner Stylight GmbH exhibits these properties and therefore was selected for the study. The subjects who took part in the study were employees of Stylight GmbH (a requirements engineer and UI expert, a software developer, two domain experts) and external test users. The subject selection reflects the parties usually involved in a software project at Stylight. The units of analysis are two software development projects. The projects were selected based on the following criteria:

- The project is a development project for an IDS-based product.
- The project is a new product development (NPD) project that involves all development stages.
- Both projects are conducted by the same development team.
- Both projects are of similar size in terms of number of features.
- The target product is business critical, i.e. it has direct impact on sales.
- The product targets consumers.

Project 1 - Quickshop. The first project was conducted with the study partner's original development process, i.e. without the suggested workflow and tool support. The project included the feature with the highest priority on the partners's road map: A "Quickshop" feature should provide the user with the possibility to add products

to his shopping cart directly from a product category overview. The feature request was elicited from surveys and focus groups prior to the case study. It is supposed to generate a higher number of products added to the shopping cart. The project duration was 11 days.

Project 2 - Checkout. The second project was based on the workflow described in Chapter 7 and the tool support Cuid. The project goal was to optimize the checkout process by reducing the steps from the shopping cart to the actual order. It was expected that this change would result in less users leaving the checkout process before completing their purchase. The project duration was 16 days.

Data Collection Procedures

The collection of data was based on first degree (interview) and second degree (usage data and work artefacts) methods. The collection of usage data was structured by using the Goal Question Metric method (GQM). The metrics are documented in detail in [3]. The collection was conducted in four steps:

Step 1: The requirements engineer was asked to assess and document all values for the metrics defined with the GQM approach after every iteration of the RE stage. This includes number and categorization of requirements with respect to the tracing status (new, unchanged, changed, removed), their names and descriptions. Also included were the number of contacts with users and co-workers and the overall effort (working hours) involved for different phases of an iteration. The documentation was form-based.

Step 2: As Project 2 also involved the use of the tool support Cuid, usage statistics created by Cuid were used in addition to the data in step 1. Cuid provided statistics for all metrics by counting artefacts, their categories and by logging toolkit usage by the different stakeholders.

Step 3: The resulting Design Prototypes have been collected and documented.

Step 4: After the projects were finalized, an interview was conducted in order to provide a second data source for the data that was recorded in steps 1-3.

Analysis Procedures

The analysis was conducted by following four steps:

- Steps 1-3 address research questions **RQ 3.2** and **RQ 3.3**
- Step 4 addresses research questions **RQ 3.1**

Step 1: In order to get a first overview of the dataset, diagrams have been created for each metric from the GQM approach. Descriptive statistics have been applied to examine frequencies, means, medians and standard deviations. The latter was used to detect outliers.

Step 2: The dataset was then reduced to aggregated views on the data after excluding outliers and insubstantial information. The resulting information was then used to build hypotheses for answering the questions from the GQM plan.

Step 3: The hypotheses and results were then discussed in a structured interview with open and closed questions. The interview was conducted with the requirements engineer.

Step 4: The usage data and the collected Design Prototypes have been compared to understand the extent to which prototyping tools were used.

Validity Procedures

There were no major threats to construct validity as the study was mostly based on second degree analysis of data which was gathered specifically for this case study. External validity and reliability were supported by precise documentation of the study environment, the execution and results. However, the case study design primarily focused on procedures for improving internal validity. This is due to the fact that threats to internal validity are manifold in this study: Different projects with different workflows and a dependency relationship between one of the experimenters and the subjects[1]. Internal validity was supported by establishing comparable project setups, i.e. both projects shared the same project team, the same tools (except for the tools and process suggested as part of this work, which was only used in the second project) and similar project properties. The dependency relationship was relaxed by introducing measurement reviews (controlling) with uninvolved colleagues who were not under supervision of the respective managing director. Learning effects have been excluded by applying the improved workflow and tool-support to the second project only.

9.3 Results

The results of the study suggest positive effects on user integration, a better detection of implicit requirements and practical integration of existing prototyping methods.

9.3.1 Case and Subject Description

Project 1 went through four RE iterations and three complete development cycles (i.e. two development cycles defaulted back to the RE phase after entering the implementation phase). *Project 2* went through two more RE iterations and contained only two complete development cycles.

[1] as mentioned before, the study was accompanied by a diploma thesis by one of the managing directors

9.3.2 How can the suggested approach be integrated with existing prototyping tools? (RQ 3.1)

As part of Project 2, both Paper Prototyping and Electronic Prototyping iterations were conducted and supported by Cuid. The Electronic Prototyping iteration was conducted with Pidoco as described in Chapter 8. The Paper Prototyping iteration, however, was conducted in a slightly unexpected way, which presents an additional way of integration with existing tool chains: The requirements engineer used a graphics application (Adobe Photoshop) to sketch mock-ups of the UI. He then printed the mock-ups and colour-coded them with the colour scheme imposed by the paper prototyping adapter Paper2Xml (cf. Section 8.2.2 and Figure 9.1). This way, both electronic prototypes with Pidoco and paper-like prototypes with Adobe Photoshop could be integrated.

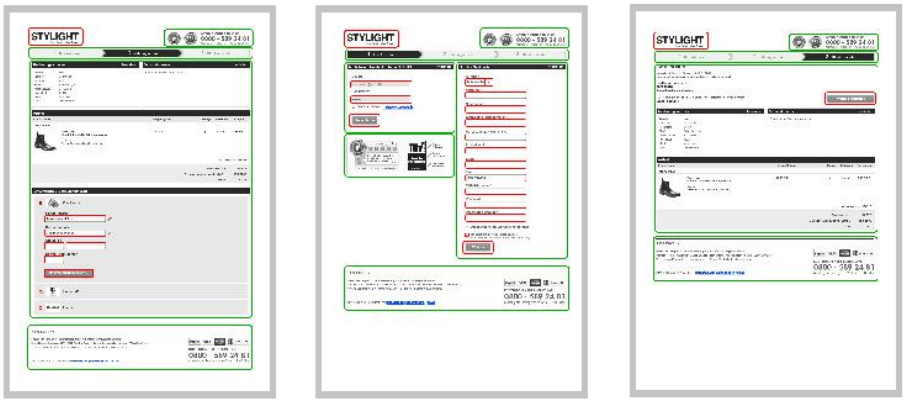

Figure 9.1: Paper-like protoypes can be created with other tools such as Adobe Photoshop.

9.3.3 What is the impact of an improved workflow and tool-support on the amount of requirements being documented and traced? (RQ 3.2)

Although the case study environment was limited in size, a number of interesting effects could be identified: Cuid was actually able to detect UI Design Patterns in the Design Prototypes of Project 2 (see Figure 9.2). As part of the post mortem interview, the requirements engineer indicated that in Project 2 more requirements were identified as compared to working without the workflow and tool support. Furthermore Cuid allowed the requirements engineer to detect a requirements violation, which otherwise would have stayed undetected until the implementation or even production phase. This is reflected by two interesting remarks by the requirements engineer:

> "I think implicit requirements were elicited that are relatively obvious in a team that works closely together. Cuid extracted them as part of the design patterns. [...]"

This closely relates to one of the findings in the exploratory case study that was described in Chapter 4. In a close working environment, responsible stakeholders

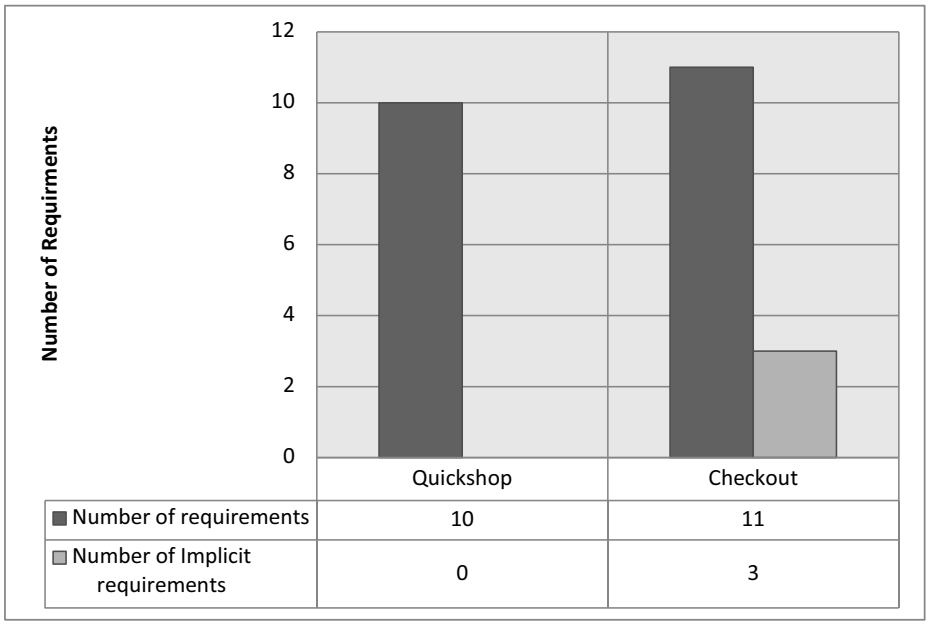

Figure 9.2: The approach helped identify requirements which would otherwise been left undocumented.

have the impression that the requirements contained in Design Prototypes are evident. Therefore, requirements specified using Design Prototypes tend to remain formally unspecified. Problems arise a) when the artefacts are to be used by other people than those who created them and b) when tracing of requirements is required.

> "An [unexpected] link [originating from the] logo and [from the] footer were identified."

Project 2 was based on the *Wizard* design pattern, which was identified by Cuid. In a subsequent iteration, additional links were detected which originated from the footer and the logo of the mockup. This violated the *Wizard* pattern and would have allowed customers to leave the checkout process at an undesired point of the conversational dialogue.

9.3.4 How does the frequency of integrating users into the design of an IDS change with an improved workflow? (RQ 3.3)

As Figure 9.3 indicates, the number of contacts with users was increased significantly in Project 2 as compared to Project 1. The requirements engineer noted during the interview that using the suggested tool chain with remote User Testing allowed to invite more users to validations. He also noted that the process itself and the structured representation helped to focus the validations more on the interaction than the visual design.

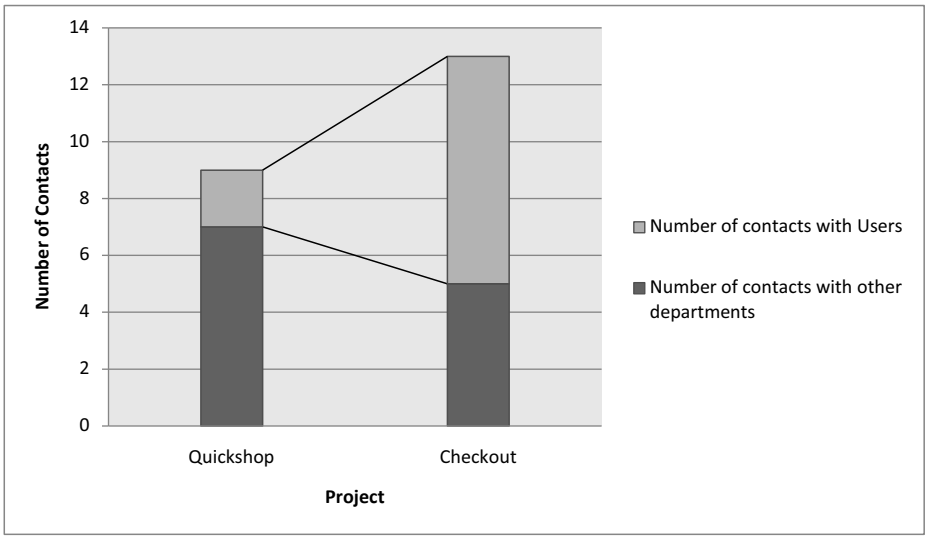

Figure 9.3: The number of contacts with users was increased due to the tool support for continuous integration.

9.3.5 Evaluation of Validity

Construct validity is given due to the specificity of the research questions: The study suggests that paper prototyping based on Adobe Photoshop and electronic prototyping based on Pidoco can be integrated with the approach (**RQ 3.1**) as it was possible to import and analyse the respective artefacts with Cuid. The amount and traceability of requirements (**RQ 3.2)** was positively affected as the usage data suggests. Triangulation (interview) has been used to clarify whether this effect can be attributed to the workflow and tool support. The same data source triangulation has also been applied to **RQ 3.3** to support construct validity. *Internal validity* was threatened by the difference of the projects, which impacted comparability, and the dependency relationship between the managing director, who helped conduct the data collection, and the subjects. Internal validity has been improved by the validity procedures described above. However, the specific results of the study (in terms of quantities measured) are still closely linked to the specific projects and tools used. The statements that were collected from the requirements engineer as part of a post-mortem interview suggest that it can be assumed that the qualitative effects in terms of integration with existing prototyping tools, improved user integration and improved requirements identification would be visible for projects with other properties (at same company) as well. *External validity* is assumed to be given by the fact that the results of the study are relevant at least for all software engineering projects which share the following properties: *1.* The goal is to develop a GUI-based IDS. *2.* The users will be consumers. *3.* The project is a new product development project which involves eliciting user-related requirements. *4.* The requirements elicitation takes several iterations and *5.* UI Prototyping is part of the requirements elicitation. *Reliability* can be achieved by applying the workflow and tool support to other projects. As the projects under investigation were conducted without direct involve-

ment of the experimenter, other researchers can repeat the case study by deploying the workflow and tool support at other companies. The limiting factor, however, is the necessity for providing adequate prototype adapters (cf. Section 8.3.1) in order to integrate the study partner's prototyping tools.

9.4 Conclusions

Summary of Conclusions

The study suggests that:

- Paper Prototyping and Electronic Prototyping can be integrated into the suggested workflow in RE projects for GUI-based IDSs.
- The suggested workflow and tool support is capable of eliciting a structured representation of Design Prototypes and to identify and trace UI Design Patterns contained in them. This helps reduce the number of complete development cycles as compared to RE without the tool support.
- The suggested workflow and tool support allows for more frequent integration of users and the integration of more users as compared to RE with milestone-oriented user integration and no toolkit support.

The suggested approach provides a more structured way of eliciting and validating requirements with users. The effort for integrating users is decreased by seamlessly integrating informal RE artefacts (Design Prototypes and UI Design Patterns) and by providing users with the necessary toolkit to take part in the process. Integrating users early on with Design Prototypes that can be experienced and representing the artefacts (and validation results) in a structured way helped improve the requirements specification. This in turn helped avoid unnecessary additional development cycles.

Relation to Existing Evidence

Besides validating the applicability of the suggested workflow and tool support to a professional software development project, the study also provides external validation to previous hypotheses: The case study described in Chapter 4 suggested that Design Prototypes contain implicit requirements that are hard to extract. The results from Project 1 suggests that RE in experienced teams in a realistic environment tends to be conducted informally, leaving the interpretation of Design Prototypes with respect to contained interactions, UI Design Patterns and validation results to the team members. Project 2 on the other hand, which used a structured approach, profited from reduced development cycles. The case study in Chapter 6 suggests that Design Prototypes in real software development projects contain implicit requirements. This case study suggests that such requirements can be identified and traced in a professional project.

Impact/Implications

The results of the study suggest that RE for IDSs profits from the continuous and seamless integration of users by eliciting and validating user-related requirements in a structured way. Thus, the approach helps to identify and avoid requirements violations which would otherwise be missed due to the informal nature of Design Prototypes and the missing integration of users.

Limitations

The suggested approach is in a very early stage. It has been validated for a small scale environment with projects ranging approx. 2-3 weeks and involving less than 20 people. It remains to be validated how the approach affects larger projects and how well it integrates with later stages of software engineering processes. Due to the semi-formal nature of the approach it is assumed that its artefacts could be integrated even with model-based approaches as discussed in Chapter 10. Future research is also necessary to assess change management for introducing the workflow in a professional environment, which was not part of the case study.

The study of human interaction will long be an essential component of all technological systems

Don Norman

10 Summary and Outlook

Interactive dialogue systems (IDSs) have become omnipresent in our modern world. We check up-to-the-minute train schedules on the internet, we find our way to the station with online navigation services while we listen to MP3 music on our smart phones, we use the time on the train working online with our documents in the cloud from our netbook. New devices and possibilities pop up almost every day. Yet, we are also used to the phenomenon that we feel lost when applications we use daily on the web suddenly change the structure of their UI, we regularly fail at setting up video conference calls because we don't get the entire stack of technologies to work, and if we want to share photos from our last trip with friends, we have to jump through many hoops to get to our goal. To build better IDSs, requirements engineering for IDSs needs to be extended with a proper model of the structure of HCI across all layers if interaction as a means for reliably documenting and tracing user requirements.

This work contributes to a seamless integration of Design Prototypes as requirements artefacts into RE and a continuous integration of users into the elicitation and validation of requirements for IDSs. The following summarizes the key findings from the previous chapters and puts them into the overall perspective of the topic. To the end of this work, an outlook is provided, which offers a broader perspective and starting points for future research.

10.1 Summary

Today, users are often integrated into the software design process at milestones only. During requirements engineering it would be helpful to integrate users more continuously to make sure that all user-related requirements are gathered and that they are traced from the beginning. As users are usually not capable of stating their needs and requirements in a structured way, informal tools such as UI Prototyping and User Toolkits can help to integrate users in a better way.

Design Prototypes contain implicit requirements that are hard to trace. Although user integration is desired, this work argues that the informal artefacts that are generated by UI Prototyping and User Testing, are not adequate to be used in RE. The requirements contained in these artefacts are not self-evident, which means

that people who were not directly involved with the elicitation tend to miss individual requirements or misinterpret them. Documenting the requirements contained in Design Prototypes in written form, however desirable, is not practicable, as the effort needed is in conflict with the fact that Design Prototypes are created in very short time and even in parallel. Not using Design Prototypes is also not an option as these artefacts fulfil an important role in facilitating requirements elicitation with users, because they are flexible, quick, and disposable.

Design Prototypes can be mapped to a formalized representation to allow for seamless integration into RE. This work suggests a structured representation for Design Prototypes that is based on a virtual protocol model for the communication between users and IDSs. The model decomposes HCI into *physical, syntactic, semantic, conversational, task* and *goal* layers. Physically, communication only takes place on the physical layer. Logically, communication between user and IDS occurs on all layers via virtual communication channels. This representation is then used to define a structured representation for a specific type of requirements, UI Design Patterns. A case study suggests that this model can be used to map existing paper prototypes and electronic prototypes to a structured representation. This is the basis for identifying UI Design Patterns in Design Prototypes and to trace them iteratively, from one prototype medium to the next. It is therefore also a contribution to solving the issue of the formality gap between informal design artefacts and requirements engineering.

Users can be integrated more continuously using User Toolkits. This work also addresses the integration of users in terms of the workflow and the tools which are necessary for this integration. The suggested workflow is based on the general steps in the requirements engineering process and extends individual stages with activities that are focused on user integration. Users can be integrated both for the elicitation and for the validation of user-related requirements. User Toolkits provide users with adequate tools to take part without requiring expert knowledge. Cuid, a prototypical implementation of a tool support for the suggested workflow, allows user integration in UI Prototyping and User Testing. The workflow and its tool support have been validated in a professional software project. The results suggest that this work contributes an approach to the integration of users into the design of IDSs that is capable of: *a)* improving the integration of Design Prototypes and related requirements such as UI Design Patterns into RE and *b)* increasing the frequency of user integration. The result is a better understanding and a better traceability of UI-related requirements.

10.2 Outlook

The following provides an overview of several topics that have been touched as part of this work, but which deserve additional research in their own right.

Extension of the pattern formalism. The formalism that is introduced in Chapter 5 is used for the structured representation of UI Design Patterns. It is not capable

of describing arbitrary constraints with respect to Design Prototypes. Beale et al. [4] describe a *UML-based approach* to specifying UI Design Patterns. This specification notation could be used to extend the formalism to cover a larger set of structural constraints. A logical next step would be to extend the formalism to *timed constraints* for expressing properties of the interaction with respect to sequences of messages that are exchanged between the user and the components of the reference architecture, e.g. synchronization or parallelism. Another possible extension would be *conditional constraints* that define constraints that apply under certain conditions. Conditions could be e.g. the *mode* of the system (which would need to be defined and modelled) or the *context of use* as defined in ISO 9241-110 [21]. Winter [90] introduces a formal representation of Usability requirements. Based on a formal representation of the layered architecture presented in this work, the formal representation could be used as a basis for a formal definition of UI Design Patterns. A formal representation of UI Design Patterns would also allow conducting a formal verification of UI Design Patterns in Design Prototypes.

Refinement. Design Prototypes evolve from iteration to iteration. The approach presented in this work allows for the general identification of UI Design Patterns in Design Prototypes. The FOCUS-based interface definitions in Chapter 5 would allow for an analysis of refinement relationships between different Design Prototypes. This has been examined as part of the research for this work and seems a promising next step to take. Being able to detect and analyse refinement relationships would allow for even better understanding of the changes that are applied to the different layers of the reference architecture between iterations. Based on this analysis a visualization could be conceived that allows to understand the changes that have been applied both to the Design Prototypes as well as their requirements. Hartmann et al. [35] introduce a revision tool that uses control flow diagrams to highlight changes in Design Prototypes. Using the before mentioned refinement analysis, this could be extended to visualize changes on different layers of the IDS. Another interesting extension would be to apply task model refinement to the structured representation of Design Prototypes. Using the approach outlined in [92] it would be possible to identify both structural and behavioural refinements of user tasks across UI Prototyping iterations.

Other modalities. This work focused on GUI-based IDSs. As mentioned before, systems with other modalities and even multi-modal systems can be designed with this approach. The reference architecture already describes which of the layers are modality-dependent, and which will remain unchanged for other modalities. The extension to other modalities would also be an interesting validation of the pattern formalism as it should be capable of describing patterns irrespective of the modality used.

Integration with ConQAT [18]. The Management Dashboard that was introduced as part of the architecture of Cuid, could be integrated with ConQAT's quality dashboards to allow for an integrated top-level view of the state of user-related requirements and numerous other requirements that are already part of ConQAT's underlying quality model. The integration would also allow the automatic generation

of test plans (which is supported by ConQAT). The test plans could then be executed with Cuid. In order to allow for an integration, the structured representation of UI Design Patterns and Design Prototypes could be used to extend the quality model for usability [91] with activities and attributes of the following form:

[User Interface | GlobalNavigation] ↦ - [Executing | Probability of Error]

Extension of the tool support. Cuid is a prototypical implementation that lacks much functionality a professional product would need. *Forward engineering* is one important topic that is currently not supported by the toolkit: Requirements engineers should be able to easily define UI Design Patterns and the related *InstanceSets* for a given Design Prototype.

Bibliography

[1] Bass, L., Faneuf, R., Reed, L., Mayer, N., Pellegrino, B., Reed, S., Seacord, R., Sheppard, S., and Szczur, M. R. A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. *SIGCHI Bull. 24*, 1 (1992), 32–37.

[2] Bauer, A. User Toolkit for Community-Based User Interface Design. Project Thesis, 2008.

[3] Bauer, A. Evaluation of a tool support for the continuous integration of users into the design and evaluation of interactive systems. Diploma Thesis, 2011.

[4] Beale, R., and Bordbar, B. Pattern tool support to guide interface design. In *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part II*. Springer-Verlag, Lisbon and Portugal, 2011, pp. 359–375.

[5] Blohm, I., Bretschneider, U., Leimeister, J. M., and Krcmar, H. Does Collaboration among Participants Lead to Better Ideas in IT-Based Idea Competitions? An Empirical Investigation. In *International Conference on System Sciences (HICSS), 2010 43rd Hawaii* (2010), pp. 1–10.

[6] Blohm, I., Fähling, J., Wallis, J. v., Birnkammerer, S., Leimeister, J. M., and Krcmar, H. Kundenintegration bei Softwareunternehmen: Eine empirische Bestandsaufnahme. In *Gemeinschaftsgestützte Innovationsentwicklung für Softwareunternehmen*, J. Leimeister, H. Krcmar, M. Koch, and K. Möslein, Eds. Josef Eul Verlag GmbH, 2011, pp. 39–56.

[7] Boehm, B. W. A spiral model of software development and enhancement. *Computer 21*, 5 (1988), 61–72.

[8] Boehm, B. W., and Papaccio, P. N. Understanding and controlling software costs: Software Engineering, IEEE Transactions on. *Software Engineering, IEEE Transactions on 14*, 10 (1988), 1462–1477.

[9] Booch, G., Maksimchuk, R., Engle, M., Young, B., Conallen, J., and Houston, K. *Object-oriented analysis and design with applications*, 3rd ed. The Addison-Wesley object technology series. Addison-Wesley Professional, Upper Saddle River NJ, 2007.

[10] Broy, M. Requirements Engineering for Embedded Systems. In *Workshop on Formal Design of Safety Critical Embedded Systems: FEMSys 1997, 16 - 18 April 1997, Munich, Germany*. GMD, 1997.

[11] Broy, M. The 'Grand Challenge' in Informatics: Engineering Software-Intensive Systems. *Computer 39*, 10 (2006), 72–80.

[12] Broy, M., Fleischmann, A., Islam, S., Kof, L., Lochmann, K., Leuxner, C., Penzenstadler, B., Mendez Fernandez, D., Sitou, W., and Winter, S. Towards an Integrated Approach to Requirement Engineering, 2009.

[13] Broy, M., and Stølen, K. *Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement*. Monographs in computer science. Springer, New York, 2001.

[14] Buxton, B. *Sketching user experiences: Getting the design right and the right design*. Morgan Kaufmann, Amsterdam, 2008.

[15] Buxton, W. Lexical and pragmatic considerations of input structures. *SIGGRAPH Computer Graphics 17*, 1 (1983), 31–37.

[16] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for multi-target user interfaces: Computer-Aided Design of User Interface. *Interacting with Computers 15*, 3 (2003), 289–308.

[17] da Silva, P. P., and Paton, N. W. User interface modeling in UMLi. *IEEE Software 20*, 4 (2003), 62–69.

[18] Deissenboeck, F. Continuous Quality Control of Long-Lived Software Systems. Dissertation, 2009.

[19] Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., and Girard, J.-F. An Activity-Based Quality Model for Maintainability. In *Proceedings of the 23rd International Conference on Software Maintenance (ICSM'07)*. IEEE Computer Society Press, 2007, pp. 184–193.

[20] Deutsche Akkreditierungsstelle Technik in der TGA GmbH. Leitfaden Usability, 2009.

[21] DIN Deutsches Institut für Normung e. V. Ergonomische Anforderungen der Mensch-System-Interaktion; Teil 110: Grundsätze der Dialoggestaltung (ISO/DIS 9241-110:2004); Deutsche Fassung prEN 9241-110:2004, 2004.

[22] Dix, A. J. *Formal methods for interactive systems*. Computers and people series. Academic Press, London, 1991.

[23] Donald A. Norman. THE WAY I SEE IT: Looking back, looking forward. *interactions 17*, 6 (2010), 61–63.

[24] Dornbusch, P. Kundennahe Entwicklungswerkzeuge für den Entwurf mobiler Geschäftsprozesse. Dissertation, 2005.

[25] Ferre, X. Integration of usability techniques into the software development process. In *Bridging the gaps between software engineering and Human-Computer Interaction* (Los Alamitos Calif., 2003), R. Kazman, L. Bass, and J. Bosch, Eds., IEEE Computer Society, pp. 28–35.

[26] Foley, J. D., and van Dam, A. *Fundamentals of interactive computer graphics*. Addison-Wesley, Reading, 1982.

[27] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, 1990.

[28] Fraser, N. M. Assessment of interactive systems. In *Handbook of Standards and Resources for Spoken Language Systems* (Berlin, 1997), D. Gibbon, R. Moore, and R. Winski, Eds., Walter de Gruyter, pp. 564–614.

[29] Gamma, E. *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley, Amsterdam, 1994.

[30] Glushko, R. Information System and Service Design: Strategy, Models, and Methods: Iteration, 2008.

[31] Green, M. Report on Dialogue Specification Tools. In *User Interface Management Systems*, G. E. Pfaff, Ed., Eurographic seminars. Springer-Verlag, Berlin, 1985, pp. 9–20.

[32] Green, M. The University of Alberta User Interface Management System. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. ACM, 1985, pp. 205–213.

[33] Greenberg, S., and Buxton, B. Usability Evaluation Considered Harmful (Some of the Time). In *CHI 2008 Proceedings* (2008), ACM Press, Ed., pp. 111–120.

[34] Gross, M. Conception and implementation of a ranking component for an existing web-based user interface design toolkit. Project Thesis, 2009.

[35] Hartmann, B., Follmer, S., Ricciardi, A., Cardenas, T., and Klemmer, S. R. d.note: revising user interfaces through change tracking, annotations, and alternatives. In *Proceedings of the 28th international conference on Human factors in computing systems*. ACM, Atlanta, 2010, pp. 493–502.

[36] Herzberg, D., and Broy, M. Modeling layered distributed communication systems. *Formal Aspects of Computing 17*, 1 (2005), 1–18.

[37] Hewett, T. *ACM SIGCHI curricula for human-computer interaction*. Association for Computing Machinery, New York, 1992.

[38] Hippel, E. v. *Democratizing innovation*, 1st ed. MIT Press, Cambridge Mass., 2005.

[39] Hippel, E. v., and Katz, R. Shifting innovation to users via toolkits. *Management Science 48* (2002), 821–833.

[40] Hosseini-Khayat, A., Hellmann, T. D., and Maurer, F. Distributed and Automated Usability Testing of Low-Fidelity Prototypes. *AGILE Conference, 2010* (2010), 59–66.

[41] Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. Papier-Mâché: Toolkit support for tangible interaction. *Proceedings of the ACM conference on human factors in computing systems (CHI 2004), Vienna, Austria, April* (2004).

[42] Krasner, G. E., and Pope, S. T. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming 1*, 3 (1988), 26–49.

[43] Krug, S. *Don't Make Me Think! A Common Sense Approach to Web Usability*, 2nd ed. New Riders, Berkeley Calif., 2006.

[44] Leimeister, J. M., Huber, M., Bretschneider, U., and Krcmar, H. Leveraging Crowdsourcing: Activation-Supporting Components for IT-Based Ideas Competition. *Journal of Management Information Systems 26*, 1 (2009), 197–224.

[45] Lin, J., Newman, M. W., Hong, J. I., and Landay, J. A. DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, The Hague, 2000, pp. 510–517.

[46] Ljaci, N. Integration von MockUp-Konzepten in die Spezifikation grafischer Bedienoberflächen. In *Informatiktage 2010*, Gesellschaft für Informatik, Ed. GI, Bonn, 2010, pp. 73–76.

[47] Löhr, A., and Brügge, B. Mixed-initiative dialog management for speech-based interaction with graphical user interfaces. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. ACM, Florence and Italy, 2008, pp. 979–988.

[48] McTear, M. F. Spoken Dialogue Technology: Enabling the Conversational User Interface. In *ACM Computing Surveys*, vol. 34. ACM, Inc., 2002, pp. 90–169.

[49] McTear, M. F. *Spoken Dialogue Technology: Towards the Conversational User Interface*. Springer, 2004.

[50] Meinhardt, H. J., and Beck, A. Usability im neuen V-Modell XT. In *Mensch & Computer 2005*, C. Stary, Ed. Oldenburg Verlag, München, 2005, pp. 101–110.

[51] Mizouni, R., Sinnig, D., and Khendek, F. Towards an Integrated Model for Functional and User Interface Requirements: Human-Centred Software Engineering. In *Human-Centred Software Engineering*, R. Bernhaupt, P. Forbrig, J. Gulliksen, and M. Lárusdóttir, Eds., vol. 6409 of *Lecture notes in computer science*. Springer Berlin / Heidelberg, 2010, pp. 214–221.

[52] Moran, T. P. The Command Language Grammar: a representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies 15*, 1 (1981), 3–50.

[53] Navarre, D. An Architecture and a Formal Description Technique for the Design and Implementation of Reconfigurable User Interfaces. In *Interactive Systems. Design, Specification, and Verification* (Berlin, 2008), T. C. N. Graham and P. Palanque, Eds., vol. 5136 of *Lecture notes in computer science*, Springer, pp. 208–224.

[54] Neill, C. J., and Laplante, P. A. Requirements engineering: the state of the practice. *IEEE Software 20*, 6 (2003), 40–45.

[55] Nepper, P. Spoken Dialogue Infrastructure for Location-based Services. Diploma Thesis, 2006.

[56] Nepper, P., Konrad, N., and Sandner, U. Talking media. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*. ACM, Singapore, 2007, pp. 348–350.

[57] Nepper, P., Treu, G., and Küpper, A. Adding Speech to Location-based Services. *Wireless Personal Communications 44*, 3 (2008), 245–261.

[58] Nielsen, J. A virtual protocol model for computer-human interaction. *International Journal of Man-Machine Studies 24*, 3 (1986), 301–312.

[59] Nielsen, J. *Usability Engineering*. Elsevier Science and Technology, 1994.

[60] Nielsen, J., and Desurvire, H. Comparative Design Review: An Exercise in Parallel Design. In *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*. ACM, Amsterdam, 1993, pp. 414–417.

[61] Nielsen, J., and Faber, J. M. Improving system usability through parallel design: Computer. *Computer 29*, 2 (1996), 29–35.

[62] Nielsen, J., Fernandes, T., Wagner, A., Wolf, R., and Ehrlich, K. Diversified parallel design: contrasting design approaches. In *Conference companion on Human factors in computing systems*. ACM, Boston Massachusetts, 1994, pp. 179–180.

[63] Norwood Sisson. Dialogue management reference model. *SIGCHI Bull. 18*, 2 (1986), 34–35.

[64] Paech, B., and Kohler, K. Usability Engineering integrated with Requirements Engineering. In *Bridging the gaps between software engineering and Human-Computer Interaction* (Los Alamitos Calif., 2003), R. Kazman, L. Bass, and J. Bosch, Eds., IEEE Computer Society, pp. 36–40.

[65] Potosnak, K. M., Hayes, P. J., Rosson, M. B., Schneider, M. L., and Whiteside, J. A. Classifying users: a hard look at some controversial issues. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, Boston Massachusetts, 1986, pp. 84–88.

[66] Ramdoyal, R., and Cleve, A. From Pattern-Based User Interfaces to Conceptual Schemas and Back. In *Conceptual Modeling – ER 2011*, M. Jeusfeld, L. Delcambre, and T.-W. Ling, Eds., vol. 6998 of *Lecture notes in computer science*. Springer Berlin / Heidelberg, 2011, pp. 247–260.

[67] Reinstadler, E. Motivating User Interface Design: Redesign of a Toolkit Interface Focusing on the Achievement Motive. Diploma Thesis, 2010.

[68] Richter, K. Methoden zur Unterstützung bei der Entwicklung plattformübergreifender Benutzerschnittstellen. Dissertation, 2007.

[69] Rumbaugh, J. *Object-oriented modeling and design*, 3rd ed. Prentice Hall, Englewood Cliffs NJ, 1991.

[70] Runeson, P., and Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering 14*, 2 (2009), 131–164.

[71] Sandner, U., Resatsch, F., Nepper, P., Leimeister, J. M., and Krcmar, H. News-on-the-go. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*. ACM, Singapore, 2007, pp. 361–363.

[72] Schattke, K., and Kehr, H. M. Motivation zur Open Innovation. In *Kommunikation als Erfolgsfaktor im Innovationsmanagement*, A. Zerfass, Ed. Gabler, Wiesbaden, 2009, pp. 121–140.

[73] Schmid, A. A Concept for a Community-based User Interface Design Process. Diploma Thesis, 2008.

[74] Schmidt, A. Implicit human computer interaction through context. *Personal Technologies 4*, 2 (2000), 191–199.

[75] Schmidt, A., Aidoo, K., Takaluoma, A., Tuomela, U., van Laerhoven, K., and van de Velde, W. Advanced Interaction in Context. In *Handheld and Ubiquitous Computing*, H.-W. Gellersen, Ed., vol. 1707 of *Lecture notes in computer science*. Springer Berlin / Heidelberg, 1999, pp. 89–101.

[76] Schnelle, D., and Lyardet, F. Voice User Interface Design Patterns. In *Proceedings of 11th European Conference on Pattern Languages of Programs (EuroPlop 2006)*, U. Zdun and L. Hvatum, Eds. Univ.-Verl. Konstanz, Konstanz, 2006.

[77] Schraml, C. Semi-automatic Conversion of a Paper Prototype to an XML Document. Bachelor's Thesis, 2010.

[78] Shneiderman, B., and Plaisant, C. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley, 2009.

[79] Sikora, E., Tenbergen, B., and Pohl, K. Modellbasiertes Requirements Engineering – Eine Situationsanalyse zum Stand der Praxis. *Softwaretechnik-Trends 30*, 1 (2010).

[80] Sommerville, I. *Software Engineering*, 9th ed., internat. ed. Pearson, Boston Mass., 2011.

[81] Spool, J., and Schroeder, W., Eds. *Testing web sites: Five users is nowhere near enough: Conference on Human Factors in Computing Systems* (2001).

[82] Szekely, P. User Interface Prototyping: Tools and Techniques. In *Software Engineering and Human-Computer Interaction*, R. Taylor and J. Coutaz, Eds., vol. 896 of *Lecture notes in computer science*. Springer Berlin / Heidelberg, 1995, pp. 76–92.

[83] Thimbleby, H. *Press on: Principles of interaction programming*. MIT Press, Cambridge and Mass., 2007.

[84] Thomas, J. J., and Hamlin, G. Graphical input interaction technique (GIIT). *SIGGRAPH Computer Graphics 17*, 1 (1983), 5–30.

[85] Tidwell, J. *Designing Interfaces: Patterns for Effective Interaction Design*, 1st ed. O'Reilly, 2005.

[86] Virzi, R. A. Refining the test phase of usability evaluation: How many subjects is enough? *Human Factors: The Journal of the Human Factors and Ergonomics Society 34*, 4 (1992), 457–468.

[87] Vukelja, L., Müller, L., and Opwis, K. Are engineers condemned to design? A survey on software engineering and UI design in Switzerland. In *Human-Computer Interaction - INTERACT 2007*, C. Baranauskas, P. Palanque, J. Abascal, and S. D. Junqueira Barbosa, Eds., vol. 4663 of *Lecture notes in computer science*. Springer, Berlin, 2007, pp. 555–568.

[88] Wagner, S., Lochmann, K., Winter, S., Goeb, A., Klaes, M., and Nunnenmacher, S. Software Quality Models in Practice: Survey Results, 2010.

[89] Wichansky, A. M. Usability testing in 2000 and beyond. *Ergonomics 43*, 7 (2000), 998–1006.

[90] Winter, S. Modellbasierte Analyse von Nutzerschnittstellen. Dissertation, 2009.

[91] Winter, S., Wagner, S., and Deissenboeck, F. A comprehensive model of usability. *Engineering Interactive Systems* (2007), 106–122.

[92] Wurdel, M., Sinnig, D., and Forbrig, P. Task Model Refinement with Meta Operators. In *Interactive Systems. Design, Specification, and Verification* (Berlin, 2008), T. C. N. Graham and P. Palanque, Eds., vol. 5136 of *Lecture notes in computer science*, Springer, pp. 300–305.

[93] Zimmermann, H. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications 28*, 4 (1980), 425–432.

A Data Model

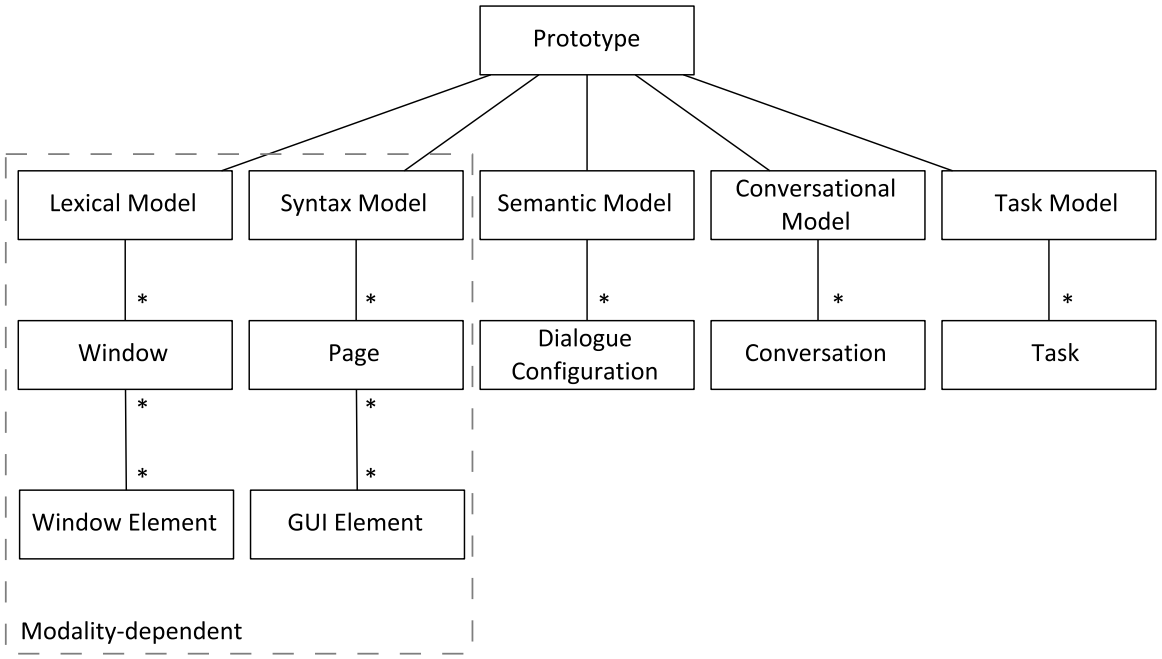

Figure A.1: The data model reflects the elements of graphical Design Prototypes.

$Objects = \{T; C; D; P; E; A; W; WE\}$

The elements of $Objects$ are named $ObjectType$s. Their elements will be called *typed objects*. Each typed object $obj \in ObjectType$ with $ObjectType \in Objects$ has a set of properties $properties.obj$. For the purpose of this work we assume:

Let $obj \in ObjectType$ with $ObjectType \in Objects$:

$$properties.obj \in \wp\{(n, v) \mid n \in PropertyNames_{ObjectType}, v \in PropertyValues\}$$

$$\forall ObjectType \in \{T; C; D; P; E; A\} : PropertyNames_{ObjectType} = \{\}$$
$$PropertyNames_W = \{x_1, x_2, y_1, y_2\}$$
$$PropertyNames_{WE} = \{x_1, x_2, y_1, y_2, clickable\}$$

$$
\begin{aligned}
PropertyNames &= \bigcup_{ObjectType \in Objects} PropertyNames_{ObjectType} \\
PropertyValues &= \bigcup_{n \in PropertyNames} PropertyValues_n \\
PropertyValues_{x_1} &= \mathbb{N}_0 \\
PropertyValues_{x_2} &= \mathbb{N}_0 \\
PropertyValues_{y_1} &= \mathbb{N}_0 \\
PropertyValues_{y_2} &= \mathbb{N}_0 \\
PropertyValues_{clickable} &= \{true, false\}
\end{aligned}
$$

B Documentation of the Case Study in Chapter 4

B.1 Questionnaire for Pre-Course Survey

Advanced Topics in User Interface Design

Please fill out the following form. We will use it for the assignment of students to mini projects. Work-load will be evenly distributed across project members, so there is no need to study game theory before answering the following questions..

If there are questions, please contact me (nepper@cdtm.de).
* Required

What's your name? *
Please provide your real name. Otherwise this form will be useless :-).

CDTM term *

	1	2	3	4	5	6	
First Term at CDTM	○	○	○	○	○	○	Sixth Term at CDTM

Field(s) of Study *

- ☐ Computer Science
- ☐ Media Computer Science
- ☐ Business Administration
- ☐ Economics
- ☐ Electrical Engineering
- ☐ Other:

Favourite Topic *

- ○ Project 1: Research Web
- ○ Project 2: Spreadshirt Integration
- ○ Undecided

Prior Experience in User Interface Design *

None

Prior Experience in Design (in general terms) *

None

Prior Experience in Usability Testing *

None

Prior Experience with CDTM's Approach to Collaborative User Interface Design based on User Toolkits *

None

Prior Experience with HTML *

None

Prior Experience with Photoshop / GIMP / etc. *

None

Comments

Is there anything else you want to share with us?

Submit

Powered by Google Docs

Report Abuse - Terms of Service - Additional Terms

B.2 Questionnaire for Mid-term Survey

Advanced Topics in User Interface Design - Interview Round 1

Please fill out the following form. We will use it for further analysis of collaborative user interface design processes. If there are questions, please contact me (nepper@cdtm.de).
* Required

What's your name? *
Please provide your real name. Otherwise this form will be useless :-).

With which design method did you work in the FIRST TWO WEEKS? *
Design Phase 1 and Test Phase 1

- (●) Paper Prototyping
- ( ) Electronic Prototyping (RapidRabb.it and CUID)

How many persons from your team were actively involved in the DESIGN of the user interfaces? *

1	2	3	4	5	6
○	○	○	○	○	○

How easy was it to work together on the DESIGN? *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

How much time did it cost to DESIGN ONE screen? *
Please enter the time invested per person taking part in the design, e.g. on average every person who designed an interface spent 15min

<5min ▾

How easy was it to understand the OTHER TEAM's test plan and prototypes? *

Very easy - no problems

How many TEST USERS did you have? *

1	2	3	4	5	6	7	8	9	10

How long did it take to TEST a SINGLE person PER SINGLE task? *
Example: On average test users spent 10min to fullfil a single task.

< 5min

What did you LIKE about using the given method (paper prototyping or electronic prototyping) for your DESIGN? *

What WAS BAD about using the given method (paper prototyping or electronic prototypin for your DESIGN? *

What did you LIKE about using the given method (paper prototyping or electronic prototyping) for your USER TESTING? *

What WAS BAD about using the given method (paper prototyping or electronic prototypin for your USER TESTING? *

Comments
Is there anything else you want to share with us?

Submit

Powered by Google Docs

Report Abuse - Terms of Service - Additional Terms

B.3 Questionnaire for Final Survey

Advanced Topics in User Interface Design - Interview Round 2

Please fill out the following form. We will use it for further analysis of collaborative user interface design processes. If there are questions, please contact me (nepper@cdtm.de). The questions in this form refer to the LAST TWO WEEKS of the design and test phase.
* Required

What's your name? *
Please provide your real name. Otherwise this form will be useless :-).

With which design method did you work in the LAST TWO WEEKS? *
Design Phase 1 and Test Phase 1

- (•) Paper Prototyping
- ( ) Electronic Prototyping (RapidRabb.it and CUID)

How many persons from your team were actively involved in the DESIGN of the user interfaces? *

1	2	3	4	5	6
○	○	○	○	○	○

How easy was it to work together on the DESIGN? *

	1	2	3	4	5	
Very easy	○	○	○	○	○	Very hard

How much time did it cost to DESIGN ONE screen? *
Please enter the time invested per person taking part in the design, e.g. on average every person who designed an interface spent 15min

<5min ▾

How easy was it to understand the OTHER TEAM's test plan and prototypes? *

Very easy - no problems

How many TEST USERS did you have? *

1	2	3	4	5	6	7	8	9	10

How long did it take to TEST a SINGLE person PER SINGLE task? *
Example: On average test users spent 10min to fullfil a single task.
< 5min

What did you LIKE about using the given method (paper prototyping or electronic prototyping) for your DESIGN? *

What WAS BAD about using the given method (paper prototyping or electronic prototypin for your DESIGN? *

What did you LIKE about using the given method (paper prototyping or electronic prototyping) for your USER TESTING? *

What WAS BAD about using the given method (paper prototyping or electronic prototypin for your USER TESTING? *

Comments
Is there anything else you want to share with us?

Submit

Powered by Google Docs

Report Abuse - Terms of Service - Additional Terms

C Documentation of the Case Study in Chapter 6

Tidwell [85] introduces more than 80 UI Design Patterns which are categorized by problem areas. Each pattern consists of a *name*, a short abstract (*"what"*), descriptions of *when, why* and *how* to apply it, and illustrative *examples*. The *what* and *how* segment of a UI Design Pattern describe the properties of the pattern with respect to its visible elements and interaction schemes. The description itself is prosaic and requires human interpretation in order to be applied to a specific IDS.

C.1 Global Navigation Design Pattern

C.1.1 Example

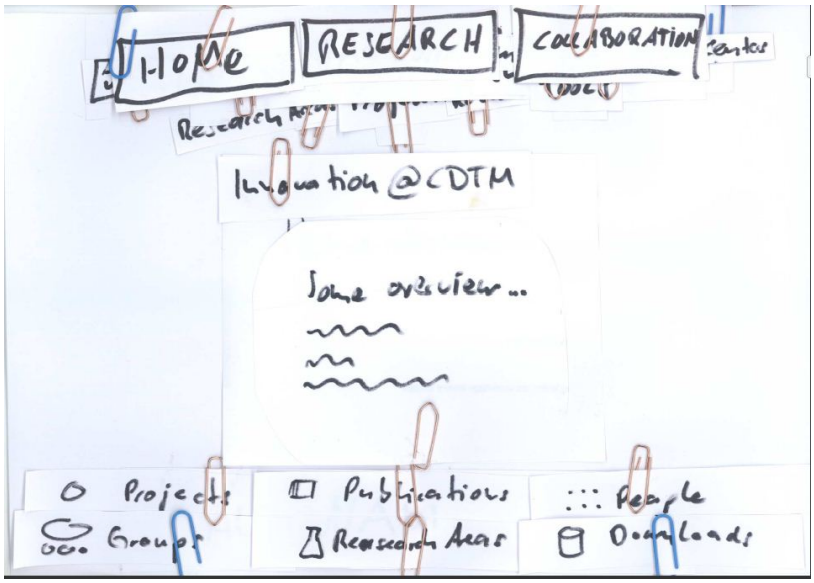

Figure C.1: Global Navigation: "Home", "Research", and "Collaboration" are present on all prototype pages and always point to the same target pages.

C.1.2 Prosaic Design Pattern

Tidwell [85] describes the Global Navigation pattern[1] as follows:

> **What:** Using a small section of every page, show a consistent set of links or buttons that take the user to key sections of the site or application.

C.1.3 Formalization

See Table C.1.

C.1.4 Heuristic

The following heuristic has been used for automatically identifying potentially matching *InstanceSets* for the Global Navigation design pattern:

1. Identify a set of conversational states ($InstanceSet_{\Sigma_1}$) that are transition targets for a large set of conversational states (($InstanceSet_{\Sigma_2}$)).
2. Reduce $InstanceSet_{\Sigma_1}$ and $InstanceSet_{\Sigma_2}$ to the maximal sub sets so that $\forall s_2 \in \Sigma_2 \exists s_1 \in \Sigma_1, a \in ActionId, d \in DialogConfigId : (s_1, a, d, s_2) \in \Delta$
3. Let $InstanceSet_{DialogConfigId}$ and $InstanceSet_{ActionId}$ of the semantic layer be the sets of $DialogConfigId$s and $ActionId$s that are inputs and outputs in the transitions that are identified in step 2.
4. Let $InstanceSet_{PageId}$ of the syntax layer be the set of $PageId$s that correspond to the dialog configurations in $InstanceSet_{DialogConfigId}$ on the semantic layer.
5. Identify the set of elements ($InstanceSet_{ElementId}$) on the syntax layer that are contained in the pages ($InstanceSet_{PageId}$) and that are mapped to the set of $ActionId$s ($InstanceSet_{ActionId}$) by f_{Syn_I}
6. Identify the set of window elements ($InstanceSet_{WindowElementId}$) on the lexical layer that correspond to the set of elements ($InstanceSet_{ElementId}$) on the syntax layer.
7. Reduce all sets so that all window elements in the resulting $InstanceSet_{WindowElemenentId}$ are aligned in either a row or a column on the respective window.

[1] citing the *What* section of the pattern. The *Use when, Why* and *How* sections, as well as other patterns are also available as a preview at http://designinginterfaces.com/firstedition/

Statement	**Layer**	*Constraint* **and** *InstanceSets*
"Using a small section of every page, show a consistent set of links or buttons that take the user to key sections of the site or application."	Conversation Layer	$Constraint^{\Delta} = \{Constraint^{\Delta}_{\Sigma_1}, Constraint^{\Delta}_{I}, Constraint^{\Delta}_{\Sigma_2}, Constraint^{\Delta}_{O}\}$ $Constraint^{\Delta}_{\Sigma_1} = (\forall, \lvert InstanceSet_{\Sigma_1}\rvert)$ $Constraint^{\Delta}_{I} = (\exists, \bot)$ $Constraint^{\Delta}_{\Sigma_2} = (\forall, \lvert InstanceSet_{\Sigma_2}\rvert)$ $Constraint^{\Delta}_{O} = (\exists, \bot)$
"Using a small section of every page, show a consistent set of links or buttons that take the user to key sections of the site or application."	Semantic Layer	$Constraint^{f_{Sem_O}} = \{Constraint^{f_{Sem_O}}_{DialogConfigurationId}, Constraint^{f_{Sem_O}}_{PageId}\}$ $Constraint^{f_{Sem_O}}_{DialogConfigurationId} = (\forall, \lvert InstanceSet_{DialogConfigurationId}\rvert)$ $Constraint^{f_{Sem_O}}_{PageId}\} = (\exists, \lvert InstanceSet_{PageId}\rvert)$ $Constraint^{f_{Sem_I}} = \{Constraint^{f_{Sem_I}}_{ActionId_1}, Constraint^{f_{Sem_I}}_{ActionId_2}\}$ $Constraint^{f_{Sem_I}}_{ActionId_1} = (\forall, \lvert InstanceSet_{ActionId}\rvert)$ $Constraint^{f_{Sem_I}}_{ActionId_2} = (\exists, \lvert InstanceSet_{ActionId}\rvert)$
"Using a small section of every page, show a consistent set of links or buttons that take the user to key sections of the site or application."	Syntax Layer	$Constraint^{f_{Syn_I}} = \{Constraint^{f_{Syn_I}}_{ElementId}, Constraint^{f_{Syn_I}}_{ActionId}\}$ $Constraint^{f_{Syn_I}}_{ElementId} = (\exists, \lvert InstanceSet_{ElementId}\rvert)$ $Constraint^{f_{Syn_I}}_{ActionId}\} = (\forall, \lvert InstanceSet_{ActionId}\rvert)$ $Constraint^{f_{Syn_O}} = \{Constraint^{f_{Syn_O}}_{PageId}, Constraint^{f_{Syn_O}}_{ElementId}\}$ $Constraint^{f_{Syn_O}}_{PageId}\} = (\forall, \lvert InstanceSet_{PageId}\rvert)$ $Constraint^{f_{Syn_O}}_{ElementId}\} = (\forall, \lvert InstanceSet_{ElementId}\rvert)$
"Using a small section of every page, show a consistent set of links or buttons that take the user to key sections of the site or application."	Lexical Layer	$Constraint^{f_{WE}}_{x_1} = (\forall, \lvert InstanceSet_{WE}\rvert, x, \geq, c_x)$ $Constraint^{f_{WE}}_{y_1} = (\forall, \lvert InstanceSet_{WE}\rvert, y, \geq, c_y)$ $Constraint^{f_{WE}}_{x_2} = (\forall, \lvert InstanceSet_{WE}\rvert, width, \leq, c_x + c_{width})$ $Constraint^{f_{WE}}_{y_2} = (\forall, \lvert InstanceSet_{WE}\rvert, height, \leq, c_y + c_{height})$ $Constraint^{f_{WE}}_{clickable} = (\forall, \lvert InstanceSet_{WE}\rvert, clickable, =, true)$ $Constraint^{f_{Lex_I}} = \{Constraint^{f_{Lex_I}}_{WindowElementId}, Constraint^{f_{Lex_I}}_{ElementId}\}$ $Constraint^{f_{Lex_I}}_{WindowElementId} = (\exists, \lvert InstanceSet_{WindowElementId}\rvert)$ $Constraint^{f_{Lex_I}}_{ElementId} = (\forall, \lvert InstanceSet_{ElementId}\rvert)$

Table C.1: Example of mapping the prosaic description of a design pattern to a structured representation.