Inspector: Interactive UI Specification Tool

Thomas Memmel and Harald Reiterer

Human-Computer Interaction Group, University of Konstanz
Universitätsstrasse 10, 78457 Konstanz, Germany
E-Mail: {memmel, reiterer}@inf.uni-konstanz.de
Tel: +49 (0) 7531 – 88 35 47

Abstract When the user interface should be specified, a picture is worth a thousand words, and the worst thing to do is write a natural language specification for it. Although this practice is still common, it is a challenging task to move from text-based requirements and problem-space concepts to a final UI design, and then back. Especially for user interface specification, actors must frequently switch between high-level descriptions and low-level detailed screens. In our research we found out that advanced specifications should to be made up of interconnected artefacts that have distinct levels of abstraction. With regards to the transparency and traceability of the rationale of the specification process, transitions and dependencies must be visual and traversable. For this purpose, a user interface specification method is introduced that interactively integrates interdisciplinary and informal modelling languages with different levels of fidelity of user interface prototyping. With an innovative experimental tool, we finally assemble models and design to an interactive user interface specifications.

1. Introduction

It is generally recognised by both software practitioners and Human-Computer Interaction (HCI) specialists that structured approaches are required to model, specify, and build interactive systems with high usability [1]. This structure should be reflected in the Software Development Life Cycle (SDLC). Nevertheless, in many organizations, UI design is still an accidental or opportunistic by-product and HCI methods are not sufficiently embedded in the overall SDLC. If they are integrated, their contribution remains marginal, thus reducing the expected positive impact on software quality. This reality can be explained by the fact that most Integrated Development Environments (IDEs) are inappropriate for supporting actors from different disciplines in designing interactive systems. Formal UI tools prevent many actors from taking part in collaborative design if they do not have adequate knowledge of specific terminologies. On the other hand, being too informal leads to misunderstandings and conflicts in communication with programmers. Moreover, on further examination, many tools turn out to be more focused on requirements management than on providing support in extracting requirements from user needs and translating them into good UI design. After all, despite - or perhaps precisely because of - the vast functionality of many tools, the outcome often is unsatisfactory in terms of UI design, usability and aesthetics. This is described as the *high threshold - low ceiling* phenomenon of UI tools [2]. In order to easily produce some results with reasonable efforts, an IDE should have a low threshold: the threshold with which one can obtain a reasonably good UI should be as low as possible. On the other hand, an IDE should have a high ceiling: the maximum overall performance of the IDE should be as high as possible. To these two dimensions, one usually adds a third one: *wide walls* (Fig. 1). An IDE should have walls that are as wide as possible, thus meaning that the range of possible UIs that can be obtained via the IDE should cover as much different UIs as possible.

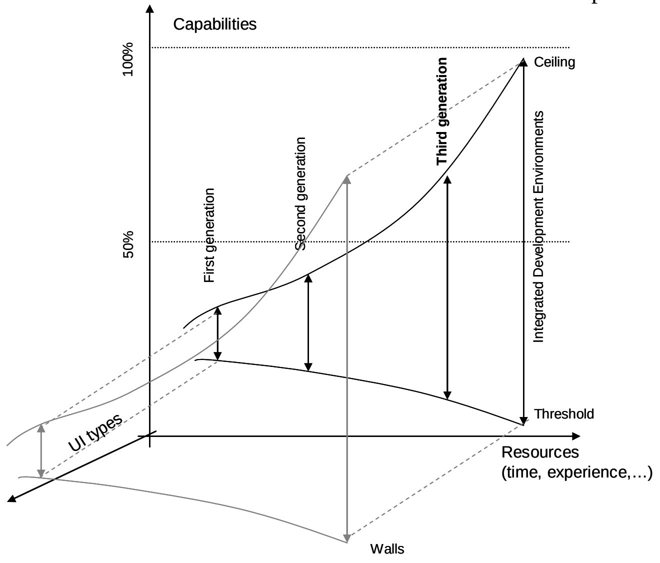

Fig. 1: Threshold vs ceiling vs walls for expressing the capabilities of IDEs

1.1 Actors in the UI specification process

Over the last 3 years, we observed UI development practice in the German automotive industry [3;4]. As a consequence of the lack of appropriate tools, many actors tend to use tools they are familiar with and which can be categorized as being *low threshold – low ceiling – narrow walls* IDEs, which has been well observed by [2]. We distinguish between two different populations of tool-users, which can be assigned to two different areas of corporate UI development projects: (1) Client: Business personnel, marketers, domain experts, or HCI experts use Office-like applications such as Word or Power Point [3] to document user needs and context of use in order to define the problem-space. They will translate the needs as analyzed, and their contextual conditions, into general usage requirements and evaluate their work at several quality gates. At this stage, responsibility is typically shared with, or completely passed on to, an IT supplier. (2) Supplier: Actors with a sophisticated IT background (e.g. programmers or designers) translate usage requirements into UI and system requirements, deliver prototypes and conclude the outcome in a UI specification. Working with UI builders, and using more formal, precise and standardized notations, they narrow the solution space towards the final UI.

1.2 Shortcomings of current UI specification practice

The difference between these groups of actors tends to result in a mixture of formats. This makes it difficult to promote concepts and creative thinking down the supply chain without media disruptions and loss of precision [3]. The following negative factors therefore contribute to UI development failure:

1. The lack of a common course of action and the use of inappropriate, incompatible terminologies and modelling languages [5] that prevent even the minimum levels of transparency, traceability and requirements-visualization that would be adequate for the problem.
2. The difficulty in switching between abstract and detailed models due to a lack of interconnectivity (compare [7]).
3. The difficulty of travelling from problem space to solution space, a difficulty that turns the overall UI development into a black-box process.
4. The burial of mission-critical information in documents that are difficult to research and have very awkward traceability. Experts are overruled when the UI design rationale is not universally available in the corresponding prototypes.
5. The perpetuation of unrecognized cross-purposes in client and supplier communication, which can lead to a premature change or reversal of UI design decisions, the implications of which will not be realized until later stages.
6. The resulting misconceptions that lead to costly change requests and iterations, which torpedo budgets and timeframes and endanger project goals.

Because of the immaturity of their UI development processes, industrial clients determined on a shift of responsibility. In our research for Dr. Ing. h. c. F. Porsche AG and Daimler AG, we found the following sticking points that tend to change current UI specification practice: (1) Due to the strategic impact of many software products, clients want to increase their UI-related competency in order to reflect corporate values by high UI quality [4]. (2) Whereas conceptual modelling, prototyping or evaluation have always been undertaken by suppliers, the client himself now wants to work in the solution space and therefore needs to develop the UI specification in-house [3]. (3) The role of the supplier becomes limited to programming the final system. The client can identify a timetable advantage from this change, and an important gain in flexibility in choosing his suppliers. Having an in-house competency in UI-related topics, the client becomes more independent and can avoid costly and time-consuming iterations with external suppliers. (4) It is nearly impossible to specify a UI with Office-like applications. The existing actors, who are nevertheless accustomed to text-based artefacts, now require new approaches. The task of learning the required modelling languages and understanding how to apply these new tools must not be an unreasonably difficult one.

1.4 Tool support that is adequate for the problem

This cultural change must be supported by an integrating UI tool that allows the translation of needs into requirements and subsequently into good UI design. In Table 1 we present a condensed overview of relevant UI tool requirements. In this paper we present both a set of models and a corresponding tool named INSPECTOR, still under development, which are designed to support interdisciplinary teams in gathering user needs, translating them into UI-related requirements, designing prototypes of different fidelity and linking the resulting artefacts to an *interactive UI specification*. The term *interactive* refers to the concept of making the process visually externalized to the greatest extent possible. This concerns both the artefacts and the medium of the UI specification itself. The latter should no longer be a text-based document, but a running simulation of how the UI should look *and* feel. Accordingly, we extend the meaning of UI prototypes to also include the provision of access to information items *below* the UI presentation layer. Being interactively connected, all of the ingredients result in a compilation of information items that are necessary to specify the UI (Table 2). In Section 2 we link our research to related work. Section 3 presents the common denominator in modelling that we developed. We explain how our tool, called INSPECTOR, will utilize the resulting interconnected hierarchy of notations. We illustrate how abstract and detailed designs can be easily created and also exported in machine-readable XML formats such as XAML or UsiXML [22].

Table 1: Requirements for UI tools for interactive UI specification; on the basis of [3;6;7]

Purpose/Added Value	Tool Requirement
Traceability of design rationale; transparency of translation of models into UI design	Switching back and forth between different (levels of) models
Smooth transition from problem-space concepts to solution space	Smooth progression between abstract and detailed representations
HCI experts can build abstract and detailed prototypes rapidly	Designing different versions of a UI is easy and quick, as is making changes to it
Provide support for design assistance and creative thinking for everybody; all kinds of actors can proactively take part in the UI specification	Concentration on a specific subset of modelling artefacts, which can be a UML-like notation or one that best leverages collaboration
The early detection of usability issues prevents costly late-cycle changes	Allowing an up-front usability evaluation of look and feel; providing feedback easily

Table 2: Main differences between prototypes and interactive UI specifications

Interactive UI Prototypes	Interactive UI Specifications
Vehicle for requirements analysis	Vehicle for requirements specification
Exclusively models the UI layer; may be inconsistent with specification and graphical notations	Allows drill down from UI to models; relates UI to requirements and vice versa
Either low-fidelity or high-fidelity	Abstract first, specification design later
Supplements text-based specification	Widely substitutes text-based specification
Design rationale saved in other documents	Incorporates design knowledge and rationale

2. Related Work

Campos and Nunes presented the tools CanonSketch and TaskSketch [7]. CanonSketch was the first tool that used canonical abstract prototypes and an UML-like notation, supplemented by a functioning HTML UI design layer. TaskSketch is a modelling tool that focuses on linking and tracing use cases, by means of which it significantly facilitates development tasks with an essential use-case notation. Altogether, TaskSketch provides three synchronized views: the participatory view uses a post-it notation to support communication with end-user and clients, the task-case view is targeted towards designers and is a digital version of index cards (well-known artefacts of usage-centred or agile developers) and the UML activity diagram view is adequate for software engineers. As we will see in this paper, we closely concur with the concepts of these tools, but our approach differs in some important areas. Firstly, and in contrast to CanonSketch, we also support detailed UI prototyping because we found that the high-fidelity externalization of design vision is especially important in corporate UI design processes. Secondly, we provide more ways of modelling. INSPECTOR integrates earlier text-based artefacts, as well as task models and interaction diagrams. Some of them are also grounded in usage-centred design, but we focused on agile models as they proved to be helpful in bridging the gaps between the disciplines (see Section 3).

The tools DAMASK [8] and DENIM [9] used a Zoomable User Interface (ZUI) approach for switching between different levels of detail through a visual drill-down process. Based on our own experience with ZUIs, we followed a consistent implementation of this technique. Calvary et al. [10] presented the CAMELEON reference framework, which proposes four levels of abstraction for UI tools: tasks and concepts, abstract UI design, detailed UI design, and the final UI. We will see that INSPECTOR supports this framework very well by the nature of the layers of abstraction used and the ZUI approach applied. However, as INSPECTOR is focused on UI specification rather than on actual UI development, it supports the final UI stage by means of UIs to other tools in the supply chain.

With respect to DAMASK and DENIM, INSPECTOR borrowed the idea of using animations to support transitions between contexts of use: when an actor needs to switch from one view to another, INSPECTOR applies a zoom-in, zoom-out technique so as to preserve continuity between the contexts of use, which has been largely demonstrated as a positive impact in SDLC [8].

3. A Common Denominator In UI-related Modelling

An advanced IDE must be able to support all actors in actively participating in the UI specification process (Table 1). This requires it to deploy modelling techniques that can be used easily by everybody. We know that the Unified Modelling Language (UML) is a weak means of modelling the UIs of interactive systems [11]. As well as its shortcomings in describing user interactions with the UI, its notation also overwhelms most actors with too much (and mostly unnecessary) detail [12]. Designing UIs is an interdisciplinary assignment and many actors might be left behind due to any formality. For instance, UML is like Office-like artefacts in being inadequate for the specification of the look *and* feel of interactive UIs.

3.1 Bridging the gaps with Agile Modelling

The identification of adequate means of modelling for UI specification is very much related to the ongoing discussion on bridging the gaps between HCI and SE. This discussion is also propelled by the very difference in the way experts from both fields prefer to express themselves in terms of formality and visual externalization. HCI and SE are recognized as professions made up of very distinct populations. In the context of corporate UI specification processes as outlined in Section 1, modelling the UI also requires the integration of the discipline of business-process modelling (BPM). The interaction layer - as interface between system and user - is the area where HCI, SE and BPM are required to collaborate in order to produce high quality UIs. As actors in corporate UI specification processes come from all three disciplines, the question is which modelling notations are adequate to extend and align their vocabulary. As we found in our previous research, agile methods are close to HCI practice [13] and therefore represent a promising pathfinder for a course of action common to all 3 disciplines. Holt [14] presents a BPM approach that is based on UML class, activity, sequence and use-case notations. Ambler based its agile version of the Rational Unified Process (RUP) on a similar, but less formal, BPM approach [15]. In general, agile approaches already exist in HCI [13], BPM [15] and SE [16] and we can define a common denominator for all three disciplines. We keep this denominator as small as possible. We filter out models that are too difficult to be understood by every actor. We do not consider models that are more commonly used to support actual implementation or that have been identified as mostly unnecessary by Agile Modelling [12, 15]. IT suppliers can deduce the structure of the UI much better from the resulting interactive specification than they can from Office-like documents.

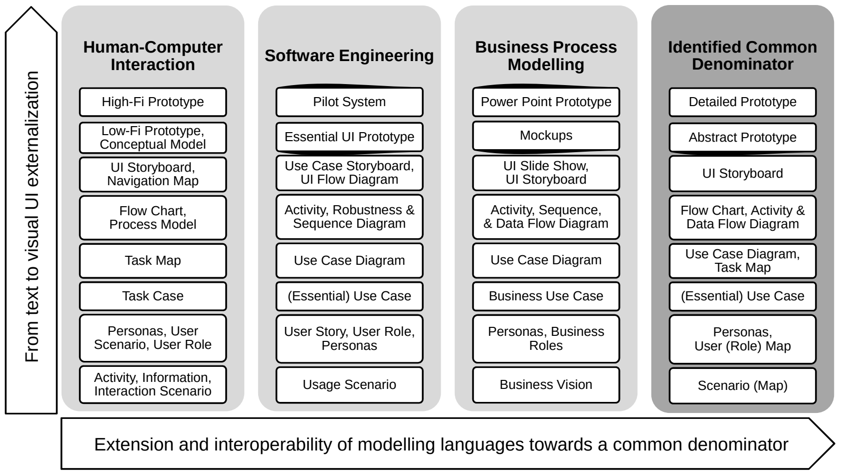
From text to visual UI externalization
Human-Computer Interaction
High-Fi Prototype
Low-Fi Prototype, Conceptual Model
UI Storyboard, Navigation Map
Flow Chart, Process Model
Task Map
Task Case
Personas, User Scenario, User Role
Activity, Information, Interaction Scenario
Software Engineering
Pilot System
Essential UI Prototype
Use Case Storyboard, UI Flow Diagram
Activity, Robustness & Sequence Diagram
Use Case Diagram
(Essential) Use Case
User Story, User Role, Personas
Usage Scenario
Business Process Modelling
Power Point Prototype
Mockups
UI Slide Show, UI Storyboard
Activity, Sequence, & Data Flow Diagram
Use Case Diagram
Business Use Case
Personas, Business Roles
Business Vision
Identified Common Denominator
Detailed Prototype
Abstract Prototype
UI Storyboard
Flow Chart, Activity & Data Flow Diagram
Use Case Diagram, Task Map
(Essential) Use Case
Personas, User (Role) Map
Scenario (Map)
Extension and interoperability of modelling languages towards a common denominator

Fig. 2: Towards a common denominator in interdisciplinary UI-related modelling

We integrate different levels of modelling abstraction to visualize the flow from initial abstract artefacts to detailed prototypes of the interaction layer. On the vertical axis in Fig. 2 we distinguish the models according to their level of abstraction. Models at the bottom are more abstract (i.e. text-based, pictorial), whereas those at upper levels become more detailed with regard to the specification of the UI. On the horizontal axis, we identify appropriate models for UI specification. Accordingly, we differentiate between the grade of formality of the models and their purpose and expressivity. The models with a comparable right to exist are arranged at the same level. At each stage we identify a common denominator for all three disciplines as a part of the thereby evolving interactive UI specification.

3.2 Text-based notations of needs and requirements: Personas and scenarios

Text-based notations can be used at any stage to document early usability attributes (usability and user experience goals, constraints, etc) with INSPECTOR's information bubbles (Fig. 3, left). For describing users and their needs, HCI recognizes user profiles, (user) scenarios [17], role models [18], and personas [19]. Roles and personas are also known in SE and BPM and are therefore appropriate for initial user-needs modelling. As an interdisciplinary modelling language, research suggests scenarios [20] - known as user stories (light-weight scenarios) in agile development [16]. In SE, scenarios – as a sequence of events triggered by the user – are generally used for requirements gathering and for model checking.

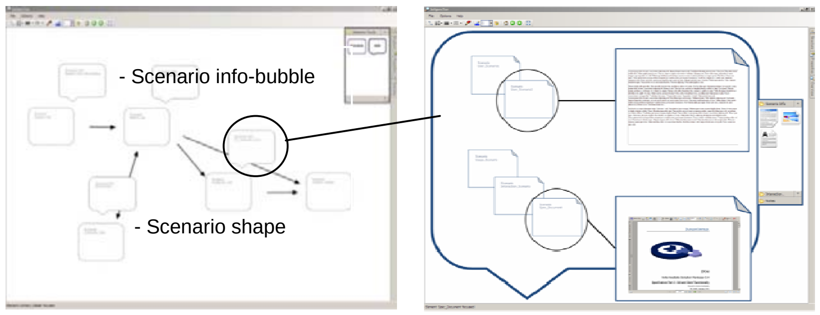

Fig. 3: Scenario map as entry stage to the modelling process (left); scenario info-bubble (right)

Such a scenario is used to identify a thread of usage for the system to be constructed and to provide a description of how the system will be used. HCI applies scenarios to describe in detail the software context, users, user roles, activities (i.e. tasks), and interaction for a certain use-case. BE uses scenario-like narrations to describe a business vision, i.e. a guess about users (customers), their activities and interests. Starting up INSPECTOR, the user can create a scenario map that relates all scenarios that will be modelled (Fig. 3, left). The user can first describe a single scenario in a bubble shape (Fig. 3, right). For this purpose, INSPECTOR provides a build-in text editor with according templates and enables the direct integration of existing requirement documents into its repository. Later, the user will zoom-in and fill the scenario shape with graphical notations and UI design (see Section 3.3ff).

3.3 Graphical notations: requirements, usage and behaviour modelling

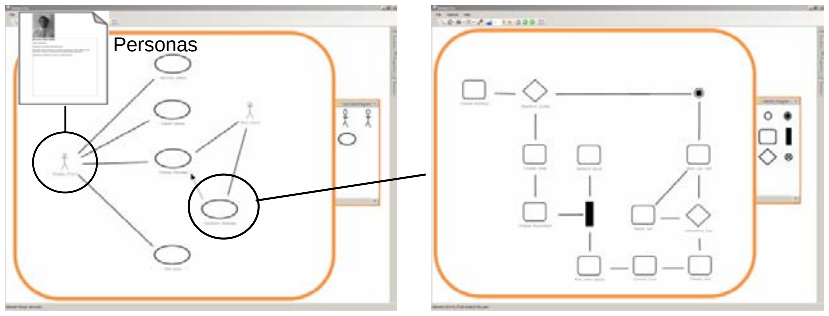

Fig. 4: Use-Case Diagram (left); Activity Diagram (right) with logic of single use case

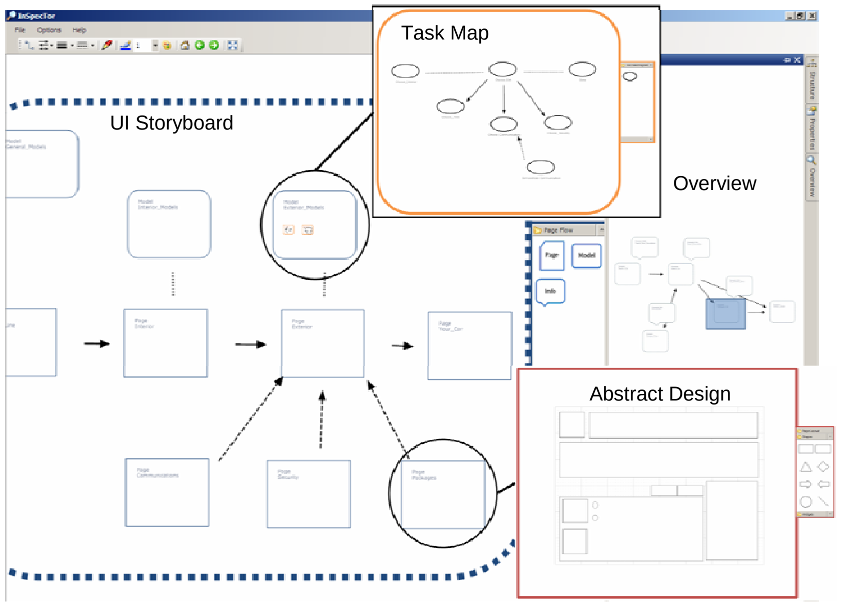

Fig. 5: UI storyboard with UI design and models; magnified areas show embedded artefacts

Entering this stage, the user needs artefacts that support the important process of translating needs into requirements. Role maps [18] help to relate created Personas to each other (Fig 4, left). Although different in name, task cases (HCI), essential-use cases (SE), and business-use cases (BPM) can be created in a classical use-case notation (Fig. 4, left). Moreover, use-case diagrams (SE, BE) overlap with use-case and task maps (HCI) [18]. The latter also help to separate more general cases from more specialized (essential) sub-cases. We considered different models for task and process modelling and, following Ambler [12;15], we again selected related modelling languages. Activity diagrams (Fig. 4, right) are typically used for business-process modelling, for modelling the logic captured by a single use-case or usage scenario, or for modelling the detailed logic of a business rule. They are the object-oriented equivalent of flow charts and data-flow diagrams. Data-flow diagrams model the flow of data through the interactive system. With a data-flow diagram, actors can visualize how the UI will operate depending on external entities. For the storyboard layer we decided to keep the typical UI storyboards we know from HCI [19]. The storyboard serves as interface layer between needs and requirement models and the UI design (Fig. 5).

3.4 UI prototyping and simulation: modelling look and feel

Prototypes are already established as a bridging technique for HCI and SE [11, 21]. HCI mainly recognizes them as an artefact for iterative UI design. Avoiding risk when making decisions that are difficult to retract is a reason why prototyping is also important for business people. Accordingly, we chose prototypes as a vehicle for abstract UI modelling. They will help to design and evaluate the UI at early stages and they support traceability from models to design. Alternate designs can be maintained in the specification landscape to safeguard the design rationale. UI elements can be assembled to templates in order to ease and speed up the design process. The visually most expressive level is the high-fidelity UI prototyping layer (Fig. 6, left). It serves as the executable, interactive part of the UI specification and makes the package complete. From this point, the actor can then explore, create and change models by drilling down to the relevant area of the UI specification. Moreover, programmers can pop-up the interactive UI specification to get guidance on the required UI properties. Therefore, all UI designs that have been created can be saved in two XML formats. On the one hand, the XAML export guarantees the reusability of the specified UIs during the development by the supplier. The XAML code can, for example, be imported to MS Expression Blend (Fig. 6, right). The XAML helps to provide simulations of the UI in a web browser such as Microsoft Internet Explorer. On the other hand, as a member of the UsiXML supply chain, INSPECTOR can contribute to the early phases of needs analysis and requirements engineering. With its UI design layer, INSPECTOR can also be compared to tools such as GrafiXML [22].

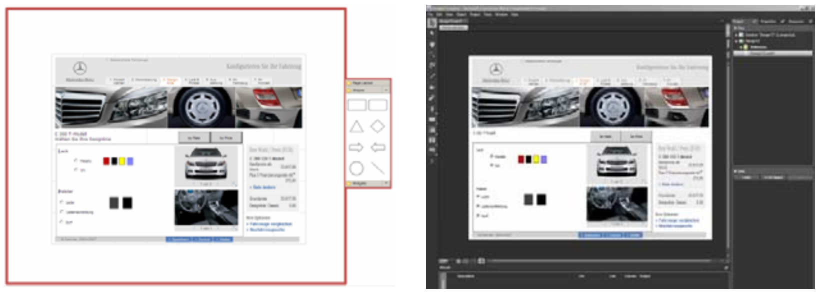

Fig. 6: INSPECTOR-made hi-fi UI design (left) in Microsoft Expression Blend (right)

3.5 Travelling through the UI specification process with INSPECTOR

INSPECTOR is based on the metaphor of a whiteboard, which is a very common tool in collaborative design environments. Basically, actors can therefore apply the models and design capabilities of INSPECTOR in arbitrary order along the UI specification process. However, the scenario map is very well suited to work on early assignments of UI specification processes. Usability and user-experience goals, business and design vision as well as reusable requirements can be captured within the information bubbles at the scenario layer. At this initial stage, problem scenarios can be textually documented. They will be enriched by concrete artefacts at the UI storyboard layer, which functions as the mediator between interconnected models and design. It encapsulates the collection of linked and interrelated artefacts by means of panning and zooming as major interaction techniques [8, 9]. This provides actors with a feeling of diving into the information space of the UI specification whiteboard. The appearance of INSPECTOR's UI is based on a linear scaling of objects (geometric zooming) and on displaying information in a way that is dependent on the scale of the objects (semantic zooming) [23]. Automatic zooming automatically organizes selected objects on the UI. Animated zooming supports the user in exploring the topology of an information space and in understanding data relationships. For switching between models and UI designs, the user can manually zoom in and out and pan the canvas. During user modelling, for example, a user shape can be linked to, and be part of, user roles, personas, and use-cases. Zooming-in on a user shape reveals more details about the underlying personas. The use-case shapes can be part of a superordinate task map and can be linked accordingly (Fig. 7). Moreover, zooming in a particular case could link to an essential use-case description and reveal more detail on user and system responsibilities. At this stage, activity and data-flow diagrams help during interaction modelling. The user can link every model to UI designs of different fidelity and vice versa (Fig. 7). During modelling, or while traversing relationships by panning and zooming, hints about the current zoom factor and the current position in the information space can be given in order to avoid disorientation. A common way of supporting the user's cognitive (i.e. spatial) map of the information space is an overview window (Fig. 5). Navigating between artefacts can be an extensive task, however, if objects are widespread in terms of being some distance along the three dimensions of the ZUI canvas. For a much faster navigation, actors can switch between artefacts with a tree-view explorer that allows a jump zoom into areas far removed from the current user focus. In order to support the assessment of the UI specification quality, we are working on a feedback component for INSPECTOR. Annotations can be attached to any canvas object. They will be used to review requirements models, to integrate results of UI evaluation studies or to incorporate notes about trade-offs or design decisions. Annotations will be accessible through a management component, which allows a direct zoom-navigation to the artefacts concerned. Equally important for design rationale, the feedback will also be stored in the UI specifications such as XAML, UsiXML.

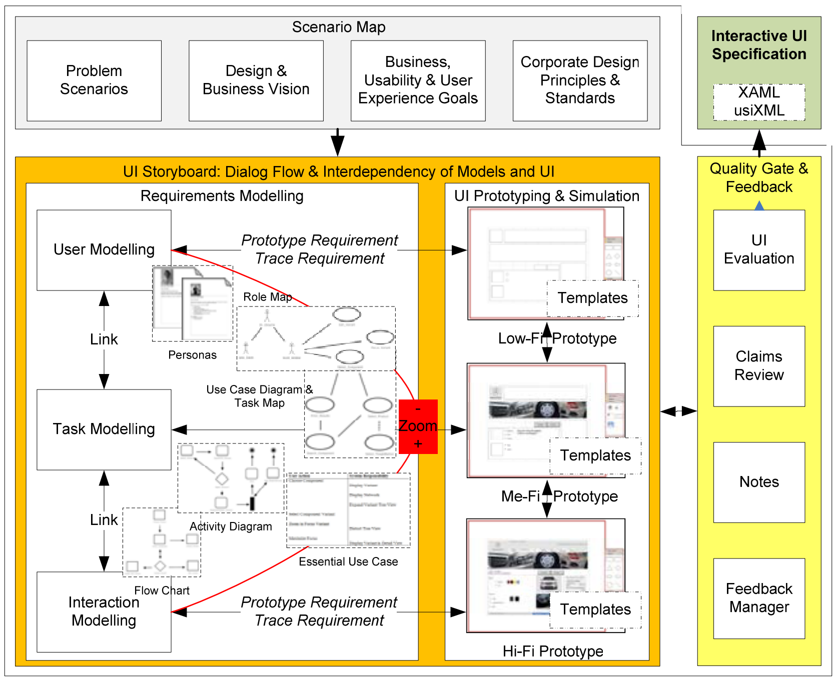

Fig. 7: Exemplified modelling and design throughput with INSPECTOR

4. Summary and Outlook

Based on our experience in UI specification and design, we have come to the conclusion that the typical methods and tools available are not adequate. With INSPECTOR, actors are supported in applying informal models they are familiar with, and are given the opportunity of UI prototyping with different fidelities. Being logically linked, transitions from abstract to detailed artefacts increase the transparency of design decisions and enhance the traceability of dependencies. This improves communication, consistency, and lastly, the necessary understanding of the overall problem space that has to be made accessible through an innovative UI. Based on a ZUI approach, INSPECTOR integrates and innovatively interconnects the required artefacts in an interactive UI specification that provides good support for roundtrip engineering at any design stage. Some evaluation studies [24] give us reasons for enhancing INSPECTOR to make it an innovative and fully capable alternative to the tool-landscape found in current industrial practice.

References

1. E. Metzker, H. Reiterer (2002): Evidence-Based Usability Engineering. In Proceedings. of the CADUI 2002: 323-336
2. P. Campos, N. Nunes (2004): Canonsketch: a User-Centered Tool for Canonical Abstract Prototyping. In Proc. of 11th International Workshop on Design, Specification

and Verification of Interactive Systems, Springer: 146-163
3. T. Memmel, C. Bock, H. Reiterer (2007): Model-driven prototyping for corporate software specification, In Proc. of the EIS’2007. Springer, 2008, to appear.
4. T. Memmel, H. Reiterer, H. Ziegler, R. Oed (2007): Visual Specification As Enhancement Of Client Authority In Designing Interactive Systems. In Proc. of the 5th Workshop of the German Chapter of the Usability Professionals Association, Weimer, Germany. In: Kerstin Roese, Henning Brau: Usability Professionals 2007, Frauenhofer IRB Verlag, Stuttgart, 99-104
5. P. Zave, M. Jackson (1997): Four Dark Corners of Requirements Engineering. ACM Transactions on Software Engineering and Methodology 6, 1 (1997) 1-30
6. N. J. Nunes, P. Campos (2004): Towards Usable Analysis, Design and Modeling Tools. In Proc. of MBUI’2004, CEUR Workshop Proceedings, Vol. 103.
7. P. Campos, N. Nunes (2006): Principles and Practice of Work Style Modeling: Sketching Design Tools. In Proc. of Human-Work Interaction Design. Springer
8. J. Lin, James A. Landay (2002): Damask: A Tool for Early-Stage Design and Prototyping of Multi-Device User Interfaces. In Proc. of the 8th International Conference on Distributed Multimedia Systems, San Francisco: 573-580
9. M. W. Newman, J. L. Jason, I. Hong, J. A. Landay (2003), DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. HCI, 18(3): 259-324
10. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, J. Vanderdonckt (2003): A Unifying Reference Framework for Multi-Target User Interfaces. In Interacting with Computer 15(3): 289–308
11. G. Sutcliffe (2005): Convergence or competition between software engineering and human computer interaction, In: Human-centered software engineering – integrating usability in the development process, Springer: 71-84
12. Scott W. Ambler (2004): The Object Primer - Agile Model-Driven Development with UML 2, Cambridge University Press
13. T. Memmel, T., F. Gundelsweiler, H. Reiterer (2007): Agile Human-Centered Software Engineering, In Proc. of the 21st BCS-HCI, 167-175
14. Jon Holt (2005): A Pragmatic Guide To Business Process Modelling, British Computer Society, United Kingdom
15. Scott W. Ambler (2002): Agile Modeling (John Wiley & Sons, NY, 2002)
16. K. Beck (1999), Extreme Programming Explained, Addison-Wesley
17. M. B. Rosson, J. M. Carroll (2002), Usability Engineering: scenario-based development of human computer interaction, Morgan Kaufmann, San Francisco
18. L. L. Constantine, L. A. D. Lockwood (1999) Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design, Addison-Wesley
19. H. Beyer, K. Holtzblatt (1998): Contextual Design: Defining Customer-Centered Systems, Morgan Kaufmann
20. S.D.J. Barbosa, M.G. Paula (2003): Interaction Modelling as a Binding Thread in the Software Development Process, In Proc. of the workshop on bridging the gaps between software engineering and human-computer interaction, Oregon, USA
21. S. Blomkvist (2005): Towards a model for bridging agile development and user-centered design. In: Human-centered software engineering – integrating usability in the development process, Springer, 219-244
22. S. Lepreux, J. Vanderdonckt, B. Michotte (2006): Visual Design of User Interfaces by (De)composition, In. Proc. of DSV-IS'2006, Vol. 4323, Springer, Berlin, 157-170.
23. Ware (2004): Information Visualization: Perception for Design, Morgan Kaufmann, San Francisco
24. T. Memmel, J. Vanderdonckt, H. Reiterer (2008): Multi-Fidelity User Interface Specifications. In Proc. of the DSV-IS 2008, to appear