First publ. in: Interactive systems - design, specification, and verification : 15th International Workshop, DSV-IS 2008 Kingston, Canada, July 16 - 18, 2008 / T. C. Nicholas Graham ... (eds.). Berlin : Springer, 2008, pp. 43-57
(= Lecture notes in computer science ; 5136)

Multi-Fidelity User Interface Specifications

Thomas Memmel[1], Jean Vanderdonckt[2], Harald Reiterer[1],

[1]Human-Computer Interaction Group, University of Konstanz,
Universitätsstrasse 10, 78457 Konstanz, Germany
[1]Belgian Laboratory of Computer-Human Interaction, Université catholique de Louvain,
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
{memmel, reiterer}@inf.uni-konstanz.de, jean.vanderdonckt@uclouvain.be

Abstract. Specifying user interfaces consists in a fundamental activity in the user interface development life cycle as it informs the subsequent steps. Good quality specifications could lead to a user interface that satisfies the user's needs. The user interface development life cycle typically involves multiple actors possessing all their own particular inputs of user interface artifacts expressed with their own formats, thus posing new constraints for integrating them into comprehensive and consistent specifications of a future user interface. This paper introduces a design technique where these actors can introduce their artifacts by sketching them in their respective input format so as to integrate them into one or multiple output formats. Each artifact can be introduced in a particular level of fidelity (ranging from low to high) and switched to an adjacent level of fidelity after appropriate refining. Refined artifacts are then captured in appropriate models stored in a model repository. In this way, co-evolutionary design of user interfaces is introduced, defined, and supported by a collaborative design tool allowing multiple inputs and multiple outputs. This design paradigm is exemplified on a case study and has been tested in an empirical study revealing how designers appreciate it.

Keywords. Collaborative design, formal and informal specifications, specification of interactive systems, usability requirements, user interface specifications.

1 Introduction and Motivations

Software practitioners and Human-Computer Interaction (HCI) specialists today concur that structured approaches are required to design, specify, and verify interactive systems [2,6,9,11,22] so as to obtain a high usability of their User Interface (UI) [19,21]. The design, the specification, and the verification of user-friendly and task-adequate UIs have become a success critical factor in many domains of activity.

In the German automotive industry for instance, a wide range of different interactive systems exists such as: in-car information systems supporting the driver while traveling, information visualization of navigation data and dynamic traffic data. Operating such systems must never compromise road safety, and the respective UIs must provide intuitive and easy-to-use navigation concepts to reduce driver's distraction to the lowest value possible. Both information visualization and navigation design are also important for corporate web sites and digital sales channels. Web applications, such as the car configuration, play an important role in the sales planning and disposal

of extra equipment. In the car manufacturers we analyzed over the past three years (among them are Dr. Ing. h.c. F. Porsche AG and Daimler AG), UI design remains a too marginal activity that deserves more attention and HCI methods are not sufficiently implied in the overall development life cycle [17,18]. Most UI development tools are inappropriate for supporting actors from different disciplines in designing interactive systems. They all possess their own particular inputs of UI artifacts expressed with their own formats and these format are generally incompatible and heterogeneous. On the one hand, *formal* UI tools may prevent some actors from taking part in collaborative design if they these tools do not have an adequate knowledge of specific input formats and terminologies. On the other hand, *informal* UI tools may lead to misunderstanding and conflicts in communication across actors, particularly with programmers. In particular, some 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 is often unsatisfactory in terms of UI design. Due to the lack of appropriate tools, many actors tend instead to use tools they are familiar with and which can be categorized as being low threshold (for application) - low ceiling (of results), a phenomenon observed in [8]. Ultimately, we distinguish two different families of tool users:

1. *Client*: actors like business personnel, marketing people, domain experts, or HCI experts use office automation applications such as word processors and presentation software [18] to document user's needs and their contexts of use [7] in order to define the problem space. They will translate the needs as perceived from the real world, and their contextual conditions, into general usage requirements and evaluate their work at several quality stages. At this stage, responsibility is typically shared with, or completely passed on to, a 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 process in a UI specification. They prefer working with UI builders, and using more formal, precise and standardized notations, they narrow the solution space towards the final UI.

1.1 Shortcomings of, and Changes Desired in Current UI Specification Practice

The difference between these two categories 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 [16]. 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 modeling languages [26] 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 [8].
3. The difficulty of traveling 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 re-

search 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 determine on a shift of responsibility and tend to change their UI specification practice:

1. Due to the strategic impact of most software, clients want to increase their UI-related competency in order to reflect corporate values by high UI quality [18].
2. Whereas conceptual modeling, 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 [16].
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 artifacts, now require new approaches. The task of learning the required modeling languages and understanding how to apply these new tools must not be an unreasonably difficult one.

1.2 Tool Support that is Adequate for the UI Design 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 (Table 1).

Table 1: Requirements for UI tools for interactive UI specification on the basis of [8,16].

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
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 modeling artifacts, 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

In this paper we present both a set of models and a corresponding tool named INSPECTOR, 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 artifacts 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 artifacts 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 modeling that we developed. We explain how our tool, called INSPECTOR, will use the resulting interconnected hierarchy of notations. We illustrate how abstract and detailed designs can easily be created and also exported in machine-readable User Interface Description Language (UIDL) such as XAML or UsiXML. Section 4 presents the results of a first experimental evaluation that highlights the contribution of our approach. Section 5 gives a summary and an outlook.

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

An early version of a model-driven UI specification method has been already presented [16]. With a separation of development concerns, different levels of abstraction and a simulation framework, we were able to establish an advanced UI modeling method. Although it was necessary to pre-define a domain-specific language (*high-threshold*), the results added significant value to a previously long-winded UI specification process (*high-ceiling*). But because the tool-chain was targeted towards the later stages of the process, office applications remained dominant during earlier phases. Moreover, the usage of a formal approach, targeted towards the generation of code from models, proved to be limiting in terms of freedom in creativity and promotion of innovative ideas. With INSPECTOR, we follow a model-based approach as our primary goal is not code generation, but the collaborative and interdisciplinary specification of non-standard UIs. However, our method and tool differ from other model-based solutions, such as the tools Vista [11], Mapper [13], and CanonSketch [8].

Vista [11] enables the designer to define mappings between four views of the same interactive system: a task model consisting of a recursive decomposition of the task into sub-tasks, a CUI model, specifications of the interaction written with the UAN notation, and specifications of the software architecture. Some of these relationships

can be established and maintained semi-automatically by Vista. No logical definition of any underlying model is made explicit. Mapper [13] explicitly establishes mappings between models, either manually or automatically, the mappings being themselves governed by a common meta-model. This system does not allow any choice of using this or that model transformation and does not provide any visualization.

CanonSketch was the first tool that used canonical abstract prototypes and an UML-like notation, supplemented by a functioning HTML UI design layer. TaskSketch [8] is a modeling 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 artifacts of user-centered or agile developers) and the UML activity diagram view is adequate for software engineers. As we will show 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 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 modeling (earlier text-based artifacts, task models and interaction diagrams).

DAMASK [14] and DENIM [21] both rely on a Zoomable User Interface (ZUI) approach for switching between different levels of fidelity through a visual drill-down process. Based on this experience and our own, we followed a consistent implementation of this technique and we chose to implement an electronic whiteboard metaphor for INSPECTOR. Whiteboards are commonly used because keeping the created artifacts visible to all actors enhances creativity, supports communication, makes it easier to achieve a common design vision and leads to faster decision-making. These tools also identified a need for supporting different levels of fidelity of requirements.

McCurdy *et al.* [15] identified five independent dimensions along which the level of fidelity could be more rigorously defined: the level of visual refinement, the breadth of functionality, the depth of functionality, the richness of interactivity, and the richness of the data model. In the remainder of this paper, the four first dimensions will be considered, the last one requiring a connection to a data model containing data. The level of fidelity is said to be *low* if the requirements representation only partially evokes the final UI without representing it in full details. Between high-fidelity (Hi-Fi) and low-fidelity (Lo-Fi), we can see *medium-fidelity* (Me-Fi). We usually observe that UI requirements only involve one representation type, i.e. one fidelity level at a time. But due to the variety of actors' inputs, several levels of fidelities could be combined together, thus leading to the concept of mixed-fidelity, such as in ProtoMixer [22]. Beyond mixed-fidelity, we introduce multi-fidelity [10] that is reached when UI requirements simultaneously involve elements belonging to different levels of fidelity, but only one level of fidelity is acted upon at a time, thus assuming that a transition is always possible between elements of different fidelity.

3 The Common Denominator in UI-related Modeling

A sophisticated UI tool must be able to support all actors in actively participating in the UI specification process (Table 1). This requires it to deploy modeling techniques

that can be used easily by everybody. We know that the Unified Modeling Language (UML) is a weak means of modeling the UIs of interactive systems [24]. 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 [1]. In most cases, moreover, designing UIs is an interdisciplinary assignment and many actors might be left behind due to the formality included in UML. Consequently, UML is like office-like artifacts in being inadequate for specifying the look *and* feel of interactive UIs. In our experience, the identification of adequate means of modeling 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, modeling the UI also requires the integration of the discipline of Business-Process Modeling (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 come from all three disciplines, the question is which modeling notations are adequate to extend and align their vocabulary.

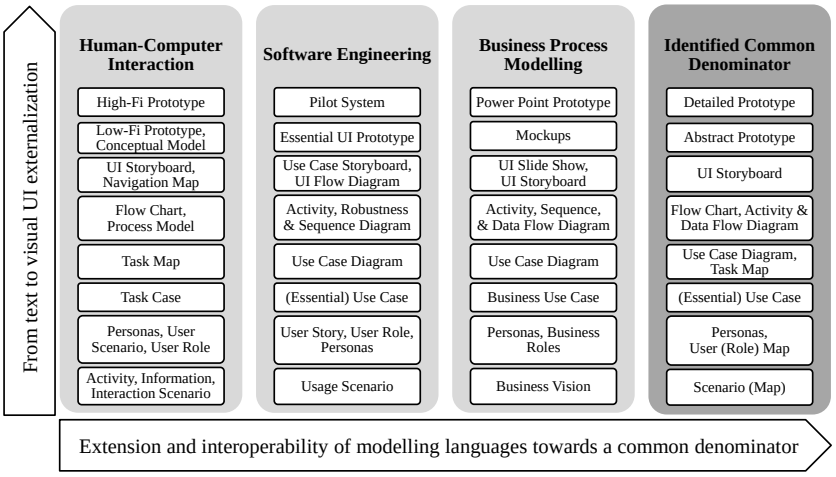

Fig. 1. Towards a common denominator in interdisciplinary modeling.

As we found in our previous research, agile methods are close to HCI practice [17] and therefore represent a promising pathfinder for a course of action common to all three disciplines. Holt [12] presents a BPM approach that is based on UML class, activity, sequence and use-case notations. Ambler based his agile version of the Rational Unified Process (RUP) on a similar, but less formal, BPM approach [1]. In general, agile approaches already exist in HCI [17], BPM [1] and SE [3] and we can define a common denominator for all three disciplines (Fig. 1). Our goal is to 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 Modeling [1]. Despite an agile freedom in terms of formality, IT suppliers can

nevertheless deduce the later structure of the UI much better from the resulting interactive UI specification than they can from Office-like documents. We integrate different levels of modeling abstraction to visualize the flow from initial abstract artifacts to detailed prototypes of the interaction layer. On the vertical axis in Fig. 1 we distinguish the models according to their level of abstraction (or level of fidelity). 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 interactive UI specification evolving thereby.

3.1 Text-based Notations of Needs and Requirements: Personas and Scenarios

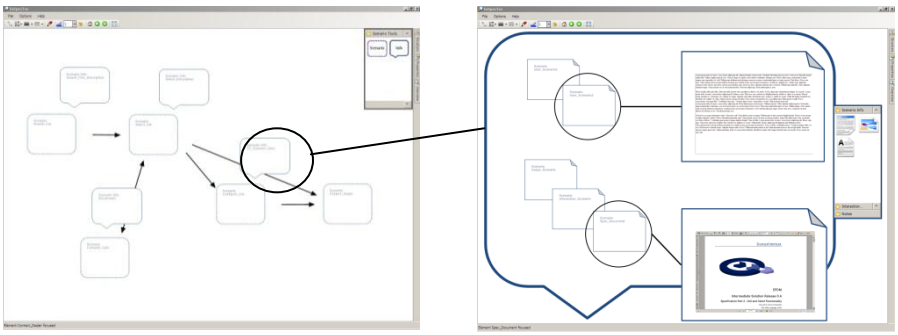

Fig. 2. Scenario map as entry stage to the modeling process (left); scenario info-bubble (right).

For describing users and their needs, HCI recognizes user profiles, (user) scenarios [23], role models [9], and personas [5]. Roles and personas are also known in SE and BPM and are therefore appropriate for initial user-needs modeling (see Fig. 1). As an interdisciplinary modeling language, research suggests scenarios [2] - known as user stories (light-weight scenarios) in agile development [3]. In SE, scenarios – as a sequence of events triggered by the user – are generally used for requirements gathering and for model checking. 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 to relate all scenarios that will be modeled (Fig. 2, left). The user can first describe a single scenario in a bubble shape (Fig. 2, right): INSPECTOR provides a build-in text editor with appropriate 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.

3.2 Graphical Notations: Requirements, Usage and Behavior Modeling

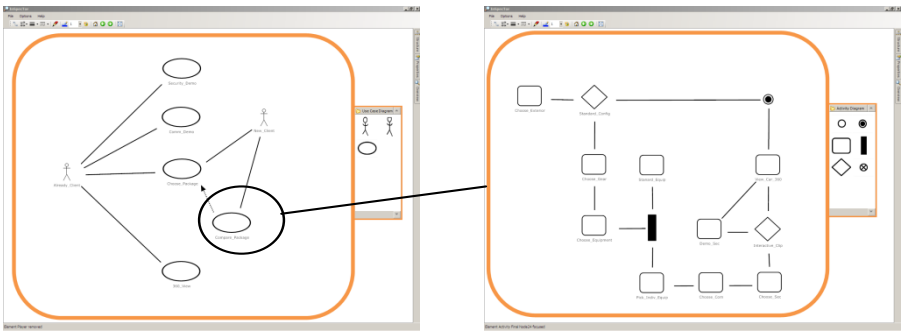

Fig. 3. Use-Case Diagram (left); Activity Diagram (right) with logic of single use case.

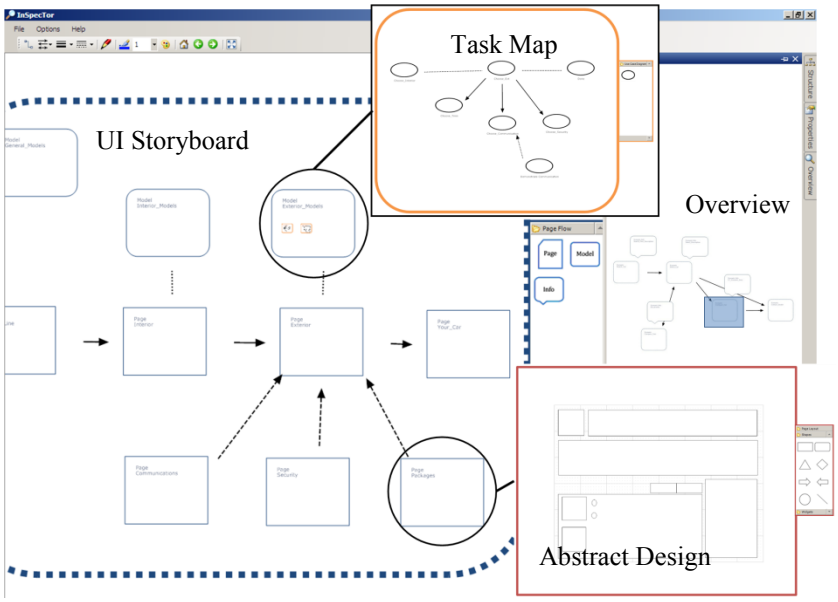

Fig. 4. UI storyboard with UI design and models (magnified areas for illustration).

Entering this stage, INSPECTOR supports the important process of translating needs into requirements (see Fig. 1). Role maps [9] help to relate user roles to each other. Although different in name, task cases (HCI), essential-use cases (SE), and business-use cases (BPM) can all be expressed in a classical use-case notation (Fig. 3, left). Moreover, use-case diagrams (SE, BE) overlap with use-case and task maps (HCI) [9]. The latter also help to separate more general cases from more specialized (essential) sub-cases. We considered different models for task and process modeling and, following [1], we again selected related modeling languages (see Fig. 1). Activity diagrams (Fig. 3, right) are typically used for business-process modeling, for modeling the logic captured by a single use-case or usage scenario, or for modeling the detailed

logic of a business rule. They are the object-oriented equivalent of flow charts and data-flow diagrams. They are more formal than the models HCI experts are usually familiar with, but they therefore extend the expert's competency in interdisciplinary modeling. Dataflow 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. Typical UI storyboards we know from HCI [18] serve as the interface layer between needs and requirement models and the UI design (Fig. 1, Fig. 4).

3.3 UI Prototyping And Simulation: Modeling Look And Feel

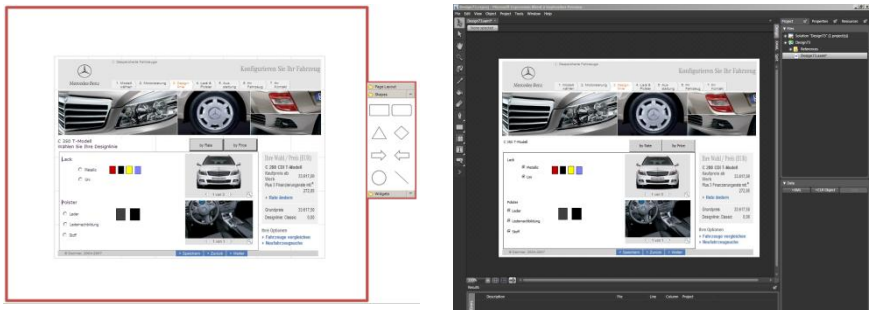

Fig. 5. INSPECTOR-made hi-fi UI design (left) in Microsoft Expression Blend (right).

Prototypes are already established as a bridging technique for HCI and SE [6,24]. HCI mainly recognizes them as an artifact 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 modeling. They will help to design and evaluate the UI at early stages and they support traceability from models to design. Alternate and competing designs as well as revised ones can all be kept in the specification landscape for later reference and for a safe-keeping of the design rationale. The visually most expressive level is the high-fidelity UI prototyping layer (Fig. 5, left). It serves as the executable, interactive part of UI specification and makes the package complete (see Fig. 1). From here on, the actor can later 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 created UI designs can be saved in two different UIDLs that are XML-compliant, thus demonstrating that INSPECTOR can accommodate any UIDL in theory. 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 Microsoft Expression Blend (Fig. 5, right). The XAML helps to provide simulations of the UI in a web browser such as Microsoft Internet Explorer. The links between pages that were created with INSPECTOR then also become links in the prototypical UI simulation. Equally important is the capability of INSPECTOR to export the results of the process in UsiXML (www.usixml.org) [13]. In this way, it can contribute to the early phases of needs analysis and requirements engineering: UI designs created can be exported from INSPECTOR and imported in any other UsiXML-compliant tool such as GrafiXML [20]. In the end, the means provided are

platform- and implementation-independent, thus making INSPECTOR compliant with the Cameleon Reference Framework [7]. Other UIDLs could be used similarly.

3.4 Feedback and Review: Creating and Managing Annotations

In order to enable actors to attach notes to artifacts in the specification space, we have added a feedback and review component. It can be used by actors to review the models and UI designs. Annotations can thus either be attached to objects on the canvas freely or be linked to specific parts of a model or page (e.g., a widget). Consistent with the ZUI interaction paradigm, the annotations can be zoomed into and accordingly provide the opportunity for editing. The annotations can also be used for giving feedback on the UI specification. When actors execute the UI simulation and explore the underlying models, they can leave notes for the UI specification team. With color coding, we distinguish the feedback provided with different grades of severity, ranging from positive ratings (green) to critical ones (red). By summarizing the reviews of actors in a management console, we can visualize conflicting artefacts, inconsistencies and any revisions that may be needed, and we can easily support a jump zoom navigation to the relevant models or UI designs.

3.5 Zoom-Based Traveling through the UI Specification Space

INSPECTOR is based on the metaphor of a whiteboard, which is a quite common tool in collaborative design environments. Because of our own experience and that of others [14,21] in developing ZUIs, INSPECTOR offers panning and zooming as major interaction techniques. In this way, it supports the principle of focus+context principle: first, the general context is identified and when it is appropriate, we can focus on some relevant part of the context, thus giving rise to a new context and so forth. It therefore provides users with a feeling of diving into the information space of the UI specification whiteboard. INSPECTOR uses [4] and the appearance of its 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) [25]. 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. Navigating between artifacts can be an extensive task, however, if objects are widespread in terms of being some distance along the three dimensions of the canvas (panning: x-axis, y-axis; zooming: z-axis). For a much faster change of focus as well as for traceability and transparency, INSPECTOR offers the possibility of creating links between models or elements of models (Fig. 7). Scenarios are the initial model, whereas the UI storyboard functions as the mediator between interconnected models and design. At early stages, 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. 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 to model the relationships of states, for example (Fig. 3).

The user can link every model to UI designs of different fidelity and vice versa. During modeling, 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. 4). In addition, INSPECTOR provides a tree-view explorer for switching between objects. This navigation support allows a jump zoom into areas far removed from the current focus.

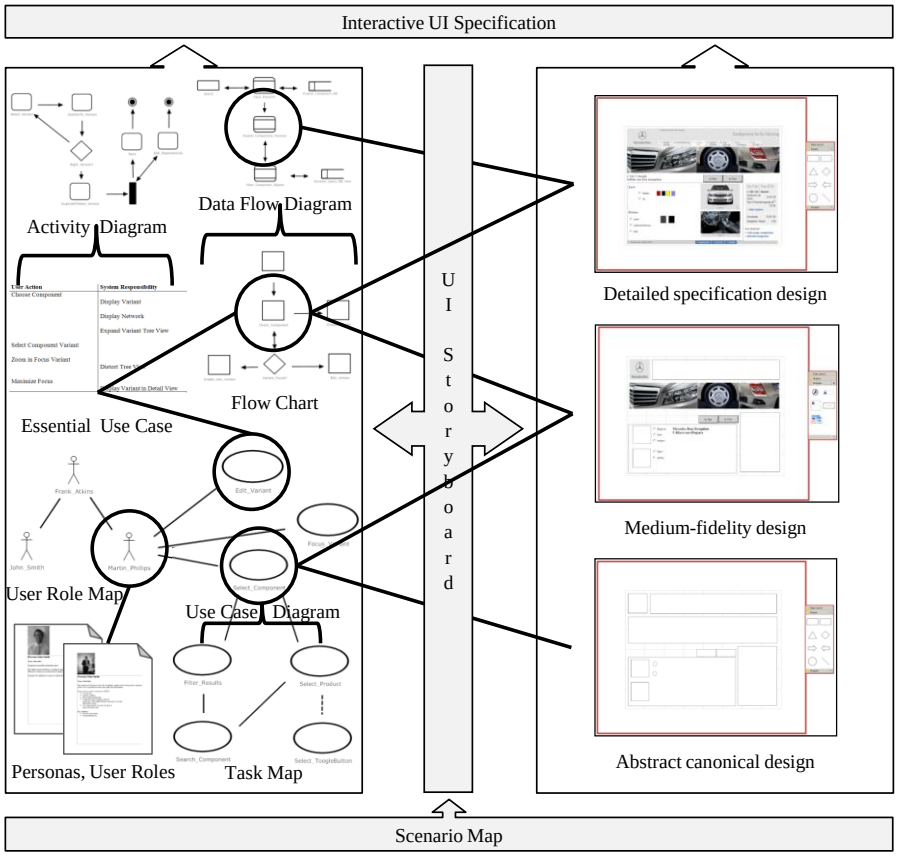

Fig. 7. Correlation of models and UI designs; exemplified modeling and design throughput.

4 Expert Feedback and Usability Study

We have started to interview software and UI specification experts (n=12) from Daimler AG in a questionnaire-based usability study. The participants were introduced to INSPECTOR through a short demonstration, a video and a supplementary text explaining the motivation for our approach. Each expert was provided with an installation of the tool and had two weeks to return his feedback by means of a questionnaire that was divided into 5 parts. The first part was designed to (1) identify the

field of activities of every respondent, (2) get an overview of the models and tools typically applied, and (3) get an assessment of difficulties along the supply chain. The second to fourth parts asked about INSPECTOR in terms of (1) the applicability of the modeling notations, (2) the completeness of the UI design capabilities and their practicability for UI evaluation, and (3) the assessment of the tool's general usability and the user experience provided. The fifth part asked if INSPECTOR could, in general, improve the UI specification practice. Currently, half of the questionnaires have been completed (n=6) and we can provide a first outline of the most important results. So far, all respondents have stated that INSPECTOR, as a tool that combines models with UI Design, contributes great value to their work style (average 4.83 pts; scale 1-5 pts). The added value was particularly identified in terms of an increased coherence of models and design artifacts, whereby INSPECTOR enhances traceability and transparency. But the study also highlighted some conceptual shortcomings. Some experts stated that during the building of a UI design, INSPECTOR could be enhanced by a contextual layer that gives the expert the chance to cross-check the design with underlying models. Instead of frequently jumping back and forth on the canvas, it should be possible to temporarily visualize models and UI concurrently. We have started to develop such a preview feature in order to further enhance the traceability of artefacts.

Other usability issues concerned the general interaction with the tool and were similar to those found during a diary study. For the latter, we used INSPECTOR in an interaction design lecture. Three groups of computer science and HCI students (n=8) were asked to use the tool during a Volkswagen use-case study on the specification of rear-seat entertainment systems. For a period of three weeks, every student wrote his own diary to give insight into (1) the kind of models created, (2) additional tools that were applied, (3) problems that occurred, (4) ratings of the user experience, (5) general issues and opinions about the tool. We decided for the diary study in order to be able to evaluate INSPECTOR over a longer period of time. Because we were interested in how the empirical results change with the duration and intensity of usage, we preferred a long-term study to classical usability tests. In weekly workshops, we discussed the intermediary results and recorded the issues for subsequent correction. By means of the diary study, we e.g. found that objects on the ZUI canvas occasionally behaved inconsistently after the tool was used for several hours and an extended amount of zoom operations had been performed. Students also reported issues with integrated external documents (PDF, Word, etc.), when they repeatedly saved and opened their projects. This led to an intensifying disarrangement of the XML structure in saved project files and significantly prevented a fluent and enduring work style. It would have been mere chance if we had identified these problems in a much shorter lab-based usability study. That way, we were able to solve these issues quickly. Moreover, we found that some participants firstly preferred to create the first abstract prototypes with paper and pencil. We realized that the use of the built-in sketching mechanism increased as soon as we provided a pen tablet as input device; like in [10]. Students were initially also not comfortable with all the notations provided and required assistance on their proper application. We addressed this issue by making a start on including a help feature that guides users through the UI specification process by explaining notations as well as their scope of application. In addition, we enhanced the affordance of templates for e.g. personas or essential-use cases to ease the understanding of the artifacts. After all, the diary study and the upgrades resulted in an im-

provement of the feedback on the tool usability: rated with an average of 1.75pts (std. 0.46) (on a 5-point Likert scale) after the first week and 3pts (std. 0.00) after the second, participants reviewed INSPECTOR with an average of 4.25pts (std. 0.46) at the end of the study. A repeated-measure ANOVA revealed a significant main effect for the rating across the weeks ($F(2,14)=105.00$, $p<0.001$). Furthermore the differences between each week are also very significant statistically (week 1 vs. week 2: $F(1,7)=58.33$, $p<0.001$; week 2 vs. week 3: $F(1,7)=58.33$, $p<0.001$).

5 Summary and Outlook

In this paper, we have introduced INSPECTOR, a collaborative design tool for sharing UI designs at various levels of fidelity in order to match the requirements that multiple actors may rely on various inputs and formats. The notion of multi-fidelity has already been proved feasible in UI prototyping [10] and is then extended to UI requirements here in a ZUI. 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. UI tools must support not only the "hard" aspects, but also the "soft" aspects of UI development to support the delivery of usable and innovative systems in the future [8]. These include support for creativity and improvisation. With our experimental tool-support, 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 artifacts 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, our INSPECTOR tool integrates and innovatively interconnects the required artifacts in an interactive UI specification that serves as a living repository of the design rationale. With our approach, we focus on actors in charge of the conceptualization, and particularly the specification, of UIs. We therefore do not support the automatic generation of the final UI like in [7], but the exchangeability of the overall specification as well as the sophisticated UI designs in machine-readable format. We will continue to enhance our tool in order to make it a fully capable and scalable alternative to the tool-landscape applied in current industrial practice.

References

1. Ambler, S.W.: Agile Modeling. John Wiley & Sons, New York (2002).
2. Barbosa, S.D.J., Paula, M.G.: Interaction Modelling as a Binding Thread in the Software Development Process. In: Proc. of the ICSE'2003 Workshop on bridging the gaps between software engineering and human-computer interaction SEHCI'2003 (Portland, May 3-4, 2003). IFIP (2003) 84–91.
3. Beck, K.: Extreme Programming Explained. Addison-Wesley, Reading (1999)
4. Bederson, B.B., Grosjean, J., Meyer, J.: Toolkit Design for Interactive Structured Graphics. IEEE Transactions on Software Engineering 30, 8 (2004) 535–546.
5. Beyer, H., Holtzblatt, K.: Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann, San Francisco (1998).
6. Blomkvist, S.: Towards a model for bridging agile development and user-centered design. In: Seffah, A., Gulliksen, J., Desmarais, M. C. (eds.), Human-centered software engineering – Integrating usability in the development process. Human-computer Interaction

Seires, Springer-Verlag, Berlin (2005) 219–244.
7. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15, 3 (2003) 289–308.
8. Campos, P., Nunes, N.: Towards useful and usable interaction design tools: CanonSketch. Interacting with Computers 19, 5-6 (2007) 597–613.
9. Constantine, L.L., Lockwood, L.A.D.: Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design. Addison-Wesley, Reading (1999).
10. Coyette, A., Kieffer, S., Vanderdonckt, J.: Multi-Fidelity Prototyping of User Interfaces. In: Proc. of Interact'2007, Springer-Verlag (2007) 149–162.
11. Elnaffar, S., Graham, N.C.: Semi-Automated Linking of User Interface Design Artifacts. In: Proc. of 3rd Int. Conf. CADUI'99. Kluwer Academic Pub. (1999) 127–138.
12. Holt, J.: A Pragmatic Guide to Business Process Modelling. British Computer Society, United Kingdom (2005).
13. Limbourg, Q., Vanderdonckt, J.: Addressing the Mapping Problem in User Interface Design with UsiXML. In: Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004. ACM Press, New York (2004) 155–163.
14. Lin, J., Landay, J.A.: Damask: A Tool for Early-Stage Design and Prototyping of Multi-Device User Interfaces. In: Proc. of the 8th Int. Conf. on Distributed Multimedia Systems, San Francisco (2002) 573–580.
15. McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B., Vera, A.: Breaking the Fidelity Barrier: An Examination of our Current Characterization of Prototypes and an Example of a Mixed-Fidelity Success. In Proc. of CHI'06, ACM Press, New York (2006) 1233–1242.
16. Memmel, T., Bock, C., Reiterer, H.: Model-driven prototyping for corporate software specification. In: Proc. of the Engineering Interactive Systems Conference EIS'2007 (Salamanca, March 22-24, 2007). M.B. Harning, J. Gulliksen (eds.). Springer-Verlag, Berlin (2007) to appear.
17. Memmel, T., Gundelsweiler, F., Reiterer, H.: Agile Human-Centered Software Engineering, In: Proc. of the 21st BCS Conf. on Human-Computer Interaction HCI'2007, 167-175.
18. Memmel, T., Reiterer, H., Ziegler, H., Oed, R.: 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. K. Roese, H. Brau (eds.): Usability Professionals 2007. Frauenhofer IRB Verlag, Stuttgart (2007) 99–104.
19. Metzker, E., Reiterer, H.: Evidence-Based Usability Engineering. In: Proc. of the 4th Int. Conf. on Computer-Aided Design of UIs CADUI'2002. Kluwer Acad. (2002) 323–336.
20. Michotte, B., Vanderdonckt, J.: GrafiXML, A Multi-Target User Interface Builder based on UsiXML. In: Proc. of 4th Int. Conf. on Autonomic and Autonomous Systems ICAS'2008 (Gosier, 16-21 March 2008). IEEE Comp. Soc. Press, Los Alamitos (2008).
21. Newman, N.W., Jason, J.L., Hong, I., Landay, J.A.: DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. J. Human-Comp. Int. 18, 3 (2003) 259-324
22. Petrie, J.N., Schneider, K.A.: Mixed-fidelity Prototyping of User Interfaces. In: Proc. of DSV-IS'2006. Springer-Verlag, Berlin (2006) 199–212.
23. Rosson, M.B., Carroll, J.M. (2002), Usability Engineering: scenario-based development of human computer interaction, Morgan Kaufmann, San Francisco
24. Sutcliffe, A.G.: Convergence or competition between software engineering and human computer interaction, In: Seffah, A., Gulliksen, J., Desmarais, M. C. (eds.), Human-centered software engineering – Integrating usability in the development process. Human-Computer Interaction Series. Springer, Berlin (2005) 71–84.
25. Ware, C.: Information Visualization: Perception for Design. Morgan Kaufmann, San Francisco (2004).
26. Zave, P., Jackson, M.: Four Dark Corners of Requirements Engineering. ACM Transactions on Software Engineering and Methodology 6, 1 (1997) 1–30.