

Interdisciplinary Visual User Interface Specification

Thomas Memmel, Florian Geyer, Johannes Rinn, Harald Reiterer
Human-Computer Interaction Lab
University of Konstanz
Universitätsstrasse 10, Box D73
Konstanz, 78465, Germany
+49 7531 88 3547

{mommel;geyer;rinn;reiterer}@inf.uni-konstanz.de

ABSTRACT

Collaborative corporate requirements engineering and UI specification demands modelling approaches that all stakeholders can understand and apply. Using traditional engineering methods, an IT organization often experiences frustrating communication and competency issues. Current processes lack adequate tool support for networked and complex projects. By integrating overview & detail and zooming, we implemented a means for supporting UI specification processes with multiple stakeholders. In contrast to existing tools, we try to bridge between the disciplines by providing related modelling languages rather than setting up more boundaries through new terminologies. Our approach beats a new path towards visual specifications of requirements and the substitution of paper-based artefacts.

Categories and Subject Descriptors

H5.2 [Information interfaces and presentation]: User Interfaces.
- Prototyping, Theory & Methods, Screen Design, Graphical User Interfaces

General Terms

Design, Human Factors, Standardization, Theory

Keywords

Zoomable User Interfaces, Visual Requirements Engineering, User Interface Specification

1. INTRODUCTION

From the authors' experience with automotive software engineering (SE), we see that the development of corporate software systems faces very demanding challenges. Several critical ingredients contribute to development failure: the increasing importance of the user interface (UI), the separation of professions, particularly SE and usability engineering (UE), and consequently a lack of methods, tools, and process models that integrate all stakeholders and their discipline-specific demands. The Standish Group identifies missing or incomplete requirements as one of the most commonly found reasons for project failure [20]. Communication among stakeholders is unsuccessful because of inappropriate terminologies and

modelling languages [16,22].

Text is ambiguous and leaves room for different interpretations. Formal models and the respective tools are generally too complex to be understood by non IT-stakeholders. After all, each stakeholder should be able to express and model his requirements in a familiar way.

Our main goal is to use specific interaction and visualization techniques to express and relate requirements models. With an innovative UI specification tool, we want to enable smooth transitions between the terminologies and models of separated domains such as SE, UE and business engineering (BE). Our motivation is to overcome recognized development issues and to bridge between usually divided disciplines. Stakeholders should be supported in actively modelling their own needs and in understanding and assessing the needs of other stakeholders. We want to integrate different levels of modelling abstraction to visualize the flow from initial abstract textual artefacts to detailed prototypes of the later interaction layer. This enables us to trace the process of translating requirements into the UI of a system and vice versa. Stakeholders can therefore track the development process and the implementation of their demands.

In Section 2 we briefly outline the results of our experience in the automotive industry. Although we present our findings by example of automotive corporate software development, we firmly believe our considerations also map software development in general. In Section 3 we present the basis for our approach, which is a hierarchy of related modelling languages. The latter will be connected to each other using a zoom-based approach described in Section 4. We explain the most important features of our specification tool in Section 5 by taking the reader through an application scenario. In Section 6 we explain the technical architecture of our tool. Finally, we summarize and conclude our research in Section 7.

2. CORPORATE UI SPECIFICATION

In the automotive industry, a wide range of different software systems exists. In-car information systems are intended to support the driver while travelling and must be intuitive and easy to use. Intranet and internet sites play an important role as well. When employees are unable to perform their tasks, the usage of intranet applications may be entirely incorrect, slow to make progress, and may finally lead to reduced acceptance and exploding support costs. Websites are important for the market success of an automotive brand and the acceptance of its products by customers. A website must create brand awareness, transport product values, enable product search, offer contact to nearby retailers and is expected to increase customer loyalty.

IT managers realize the risk of bad UI usability is considerable, and that it is an economic one. Methods of UE can significantly enhance the quality of a software product in terms of user performance and user experience. End-user participation in requirements engineering (RE) is important, but is not often

used [18]. Employing UE methods during early stages of RE often causes conflicts among stakeholders facing shrinking IT budgets and pressure of time. Collaborative design or carrying out surveys and formulating results textually are too time-consuming. Consequently, in many projects user needs and usability goals remain unaccounted for. But considering the value of corporate design (CD) and corporate identity (CI), important design decisions that affect usability, look *and* feel, are critical.

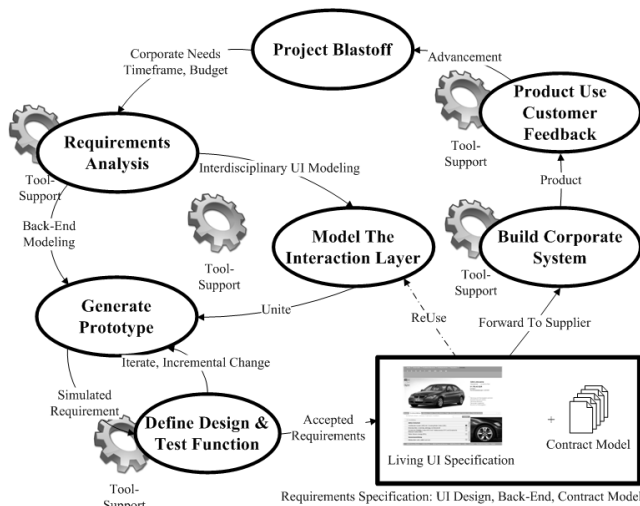


Figure 1: Tool-based UI specification process

New approaches to RE and UI specification practice are necessary. Critical parts of the system, including the UI in particular, have to be evaluated before any line of code is written by the supplier. This ensures that the later UI matches corporate values and therefore avoids design failure as well as costly late-cycle changes. Especially when the actual system implementation takes place in another country, the UI specification needs to be as complete, unambiguous and expressive as possible. Whenever the programmer is unsure about the client's demands, he should be able to pop-up a visual specification to get an impression of the desired look and feel. The supplier also benefits if the specification method previously used is compatible with its course of action. All in all, an appropriate specification process envisions a consistent usage of one tool that is used and understood by all stakeholders (see Figure 1). After all, following such a process certainly means extending the effort on the client's side. But this is the opposite of an enhancement of client-side design competency and more control of the overall process.

Currently, a lot of RE tools are available on the market [1]. Hoffmann et al. [9] define requirements for these tools and outline the need for supporting all stakeholders in actively taking part in the software design process. This requires modelling techniques that can be easily used by everybody to e.g. sketch page flows or UI mockups. Furthermore, stakeholders should be able to evaluate design solutions and submit their feedback without taking part in activities that are time-consuming for both them and developers [18]. We realized that such tools are typically hard to find and previously presented an evaluation of the promising *iRise* specification tool [12]. By separating UI prototyping from coding, end-users are able to easily take part in collaborative design. Unlike Rashid et al.'s [18] approach, end-user participation is not limited to assessing third-party design proposals. The main goal of the tool is to keep the graphical definition of requirements as simple as it is with typical office applications.

In a survey among developers in the automotive industry, we found that this approach is indeed sound. Most stakeholders actually use standard office products such as Word, Excel or PowerPoint to model their requirements [4,11]. As well as their familiarity with such applications, the most important reasons for avoiding other tools is their bad usability, the great effort needed in learning to use them, and the difficulty of understanding the related terminologies. Hence, there is a great need for a standardized and capable specification process. Later on, programmers use completely different tools and need to translate Word documents and PowerPoint slides into SE models and code. At this stage, there is a real danger of engineering failure, e.g. through misinterpretation or loss of precision. Furthermore, the black-box process of translating stakeholder requirements into a running system makes it more difficult to cross-check the result with requirements at the quality gateway. The media-disruption makes reverse engineering a laborious assignment.

Despite various advantages, *iRise* is limited to a small range of informal modelling languages and lacks mechanisms for reusability of code [12]. Although *iRise* is able to generate an interactive prototype from visual and informal UI models, the simulation is only available in compiled format. We consequently focused our recent endeavours on developing UI specification tools that are as easy to learn and use, but which offer more functionality. One of the elements in our own specification tool-chain is based on a model-driven approach [11]. We separate the different views of the overall specification into different stakeholder concerns such as design, behaviour, and content. Consequently, stakeholders are allowed to employ models of their domain to describe the system. For example, designers or even end-users can easily build UIs by using simple or advanced GUI designers. Software engineers can build process flow diagrams and develop the logic of the resulting system. The distinct parts are later combined by a domain framework and, in addition, code is generated automatically [11]. Hence, stakeholders and end-users can experience their modelling effort immediately and interactively at the push of a button.

Through the application of our tool in automotive engineering practice, we realized that such a tool-chain was badly needed and would therefore be highly appreciated and accepted by stakeholders. It is a promising basis for implementing a mature and standardized engineering process. In addition, the relatively small developments costs were a compelling motivation to continue designing new means of collaborative design for interdisciplinary processes. As described in [11], our goal was to produce tool support for visual UI specification in order to be able, as far as possible, to substitute textual and paper-based documents, which fail to map real UI behaviour. But as ordinary specification documents usually also include models that describe non-UI parts of a system, we came to the conclusion that it is also necessary to integrate them in a visual specification (see Figure 2). Hence, we want to integrate comprehensible models together with the running interactive UI simulation as a visual specification of an interactive system. These models will also be visualized interactively and offer means for navigation and manipulation. Technically, different modelling languages can be brought together e.g. with Z-based Object-Oriented Modelling (ZOOM) [17]. Research shows that this also includes the event-driven forwarding of changes to other models in order to ensure consistency. In the following sections, we can therefore concentrate on presenting techniques for a user-centered and task-adequate access to such networked models. Incidentally,

the abbreviation ZOOM points to our idea of using a Zoomable User Interface (ZUI) for this purpose.

3. DISCIPLINE-SPECIFIC MODELLING LANGUAGES

For describing non-UI related requirements, all stakeholders should employ the models they are familiar with. We therefore distinguish between three different kinds of stakeholders (see Figure 2). The business stakeholder e.g. specifies fundamental requirements and project goals using both text-based methods and process modeling notations like BPMN¹. The software engineer may also employ textual definitions at first, but typically switches to formal modelling languages earlier in the process. In order to define functional requirements, SE knows UML notations² (e.g. class diagram, activity diagram, use case diagram, state machine diagram, sequence diagram). In consequence, the UI is usually not described sufficiently. The usability engineer also has a different view on the requirements specification. To him, it is important to understand non-functional requirements such as user needs in order to be able to conceptually develop suitable interaction design. The usability engineer will document compelling design principles and define user performance as well as user experience goals. In most cases, this information is explained in text-based documents such as style guides. Otherwise, he usually creates user role and task models as well, and produces other semi-formal notations for interaction, sequence and activity diagrams.

In the end, the applied artefacts can be divided into three classifications. There are text-based methods, graphical notations and visual UI prototypes (both static and coded). In general, textual artefacts are typically applied early in a process, while more sophisticated means of expression are used during later stages. In order to show the dependencies of such models, we link them to each other in both the horizontal and vertical directions (see Figure 2). The horizontal layers are used to make similarities and dependencies more obvious. By offering adequate mechanisms of interaction control, smooth conversions on the horizontal layer should support a shared understanding across discipline-specific expertise. The vertical layers represent different visualizations of requirements expressed through models of a distinct level of abstraction. Stakeholders should be able to focus on specific problems without being deflected by unnecessary detail. After all, the UI representation at the interaction layer can be related to the corresponding models at any time. Finally, models affect each other horizontally and vertically. Changes to abstract models will influence those of other disciplines on equal or superior levels, and vice versa.

In order to bridge the disciplines as smoothly as possible, we chose to integrate modelling languages that show significant overlap (see Figure 2). Many modelling techniques are repeated across disciplines, although varying in formality and objectivity of use.

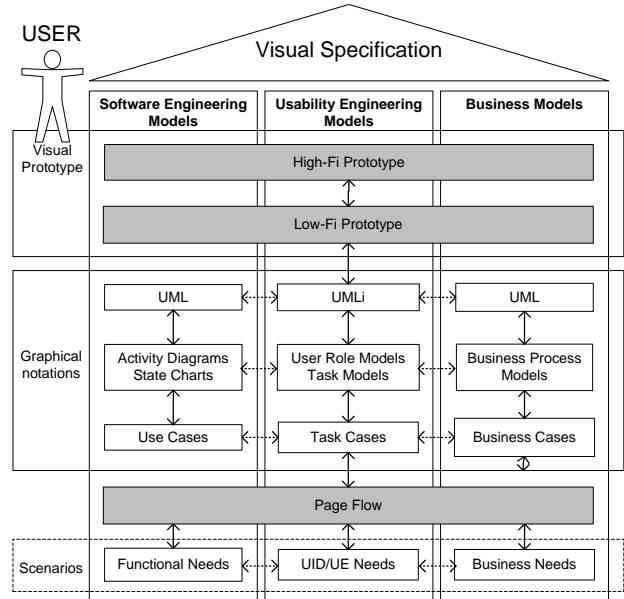


Figure 2: Linked models for visual specification (models supported by iRise highlighted in grey; dotted line indicates starting point of visual specification)

For example, Use Cases (SE) and Task Cases (UE) can easily be related to each other as well as to Business Cases. Furthermore, Task Models (UE) can be brought together with UML notations like Activity Diagrams (SE) [5]. BPMN is based on flowcharting techniques and therefore has evident correlation with UE and SE models (see Figure 2). As the mapping of modeling languages very much depends on the application domain and might vary across projects accordingly, we decided to summarize them with the general term UML(i) (see Figure 2). This underlines that our approach does generally not limit the quantity and kind of used modeling techniques.

Consequently, we do also not restrict our approach to a 1:1:1 mapping among discipline-specific models. As Figure 2 shows, it is necessary to allow a n:n:n mapping as well. Software engineers might employ state machines in addition to activity diagrams. Besides task models, usability engineers most likely also define user role models. Altogether, stakeholders sometimes prefer to integrate different kinds of discipline-specific notations for the same purpose. Hence, our approach would allow the integration of several models on the same level of abstraction - as long as it helps a better understanding of the problem at hand. After all, at least one of these models has to provide the needed interfaces to those of the other disciplines.

Nevertheless, we recommend keeping the amount of modeling languages low. This helps to avoid inconsistency of models, prevents unnecessary redundancy and prevents an overhead of distinct terminologies. Our goal is to reduce inter- and intra-discipline complexity as far as possible.

Finally, for communication and a shared understanding, all models are taken up with a detailed UI prototype at the interaction layer. Together, the prototype and the underlying models define the visual specification of the software system.

4. ZOOM-BASED SPECIFICATION APPROACH

In order to gain an overall and detailed view of the system, all stakeholders must be able to navigate between the different abstraction levels and to switch between the models. Linking up with our experience in developing ZUIs [6,8], and following the idea of Damask [10] and Denim [13], we use a ZUI to

¹ See <http://www.bpmn.org/>

² See <http://www.omg.org>

intuitively map the dependencies and transitions of different models.

Damask and Denim were implemented to support web designers in early-stage informal UI design and cross-platform prototyping. Accordingly, the tools e.g. allow sketching UIs with a pencil tool. As stated in Section 2, our goal is to support various stakeholders in producing high-detail UI specifications. Hence, our approach differs in terms of formality and the kind of modeling languages embedded. Nevertheless, our idea can be said to be an enhancement of Damask and Denim, as we reuse many of their concepts, like e.g. the zoom-slider (see Section 5).

ZUIs offer *panning* and *zooming* as the main techniques for direct interaction and navigation [14]. The appearance of information in ZUIs is mainly based on a linear scaling of objects (*geometric zooming*) and displaying information in a way that is dependent on the scale of the objects (*semantic zooming*). Various other zoom-techniques support interaction with ZUIs. For example, *automatic zooming* organizes selected objects automatically at the centre of the UI [7]. *Animated zooming* supports the user in exploring the topology of an information landscape and in understanding data relations [2]. Hints about the current zoom factor and the current position in the information space can be given in order to avoid disorientation [15]. A common way of supporting the user's cognitive map of the information space is an overview window. Users can develop spatial maps faster in those environments where they can quickly and simply view the entire information space [21].

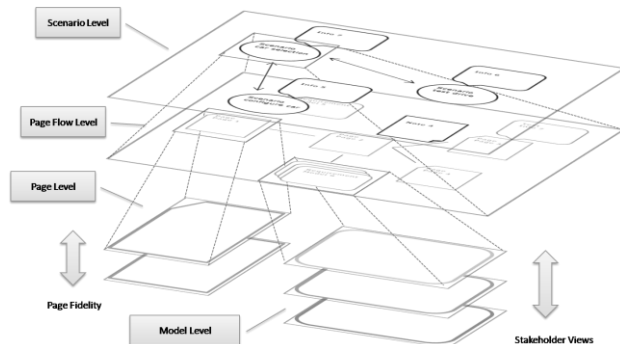


Figure 3: ZUI layers of the information landscape

ZUIs simplify the illustration of hierarchies by the representation in different levels of scaling and enlargement. This advantage is very appropriate for visualizing the hierarchy of interdependent modelling languages as illustrated in Figure 2. In order to apply the techniques and concepts of ZUIs to the domain of visual UI specification, we created appropriate graphical notations for the corresponding models to represent them on the canvas. Complexity of models is substantially facilitated by hierarchical separation in the information landscape (see Figure 3).

We divide the information landscape into several layers. The *scenario level* gives an overview of all usage scenarios that were modelled with the tool. The *page flow level* represents a scenario in detail by showing the corresponding pages and the transitions between them. From the *page flow level*, the user can zoom into the detail of a single page to design its UI. Alternatively, the user can choose to model the requirements with the various graphical notations available (see Section 3).

5. APPLICATION SCENARIO

In order to describe the layering and hence the utilization of our tool in detail, we present an example of the development of a web-based car configurator.

We use scenarios as an entry point for exploring the visual UI specification (see Figure 2). Hence, they represent the most abstract specification of requirements. Examples for scenarios of a car configurator are the “selection of a car” and the “configuration of a car”. Scenarios are a means of communication known by all disciplines [19]. Thus, at this level, we do indeed distinguish between discipline-specific views. Each scenario is explained by a textual description of underlying functional, non-functional and business needs. These descriptions are attached to scenario models by balloon symbols (see Figure 4, 1). The scenarios are composed by dragging an element from the toolbox (see Figure 4, 3) and dropping it on the canvas. The user has to click on a description (i.e. info) element for editing.

In the centre of the application window is the zoomable information space where users can place their different models and artefacts (see Figure 4, 1). Next to the main canvas we integrated the overview & detail view (see Figure 4, 2). The overview window supports understanding and navigating the information space [21]. As stated before, it supports users' orientation and helps them to browse to related elements on the screen. Within the overview, the visible range is always highlighted by a blue rectangle that is regularly updated (see Section 6 for technical detail). When selecting an object in the overview window, the focus is also set on the corresponding element on the main canvas. Hence, a change of the focus through a zoom-operation updates both views (see Figures 4 and 5).

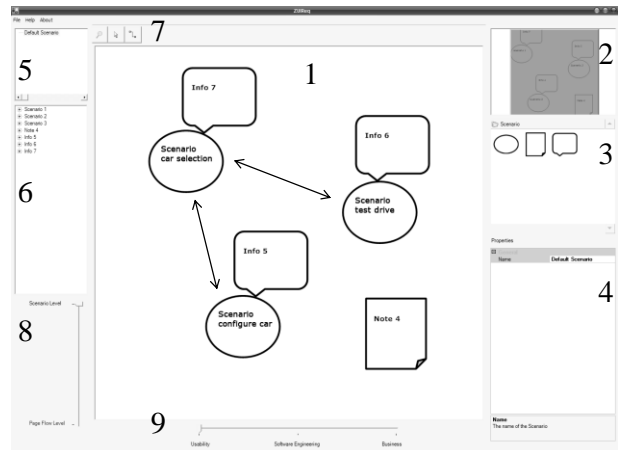


Figure 4: Main window with scenario level

Below the overview canvas, the user finds a toolbox that offers various elements for creating models for his requirements on the main canvas (see Figure 4, 3). The slider at the bottom is used to switch between discipline-specific views on the requirements (see Figure 4, 9). At all levels, we distinguish between functional, usability and business needs. Here, the information in the balloon elements is varying accordingly. When a user switches his role to one from another discipline, the toolbox offers different objects for modelling requirements at a specific level of abstraction. This adaptation of the UI is adequate for the problem and helps to prevent an inconsistent mixture of modelling languages. Furthermore, this ensures that users are able to change the discipline-specific view to related modelling languages at the same horizontal layer (see Figure 2). The objects can be placed on the canvas by drag and drop.

On the upper-left side of the application window the user finds a tree component for switching between several scenarios (see Figure 4, 5). Below that there is another tree view that represents the overall hierarchy of objects contained on the current canvas (see Figure 4, 6). Both tree views serve as navigation support. They help to return from e.g. having zoomed into an information bubble (see Figure 5).



Figure 5: Zoom-in on a description bubble of a scenario

In particular, however, they also allow a jump zoom into areas of the information space far removed from the current user focus [15]. This could either be other scenarios (*scenario level*) or objects in the same page flow (*page flow level*). The corresponding zoom-operation is triggered by a click on a tree node.

A toolbar on the top of the window is available for interaction with the canvas (see Figure 4, 7). By choosing one of the buttons, the user changes the interaction mode to selecting, moving or zooming objects. If the user selects an object in the ZUI canvas by a single mouse-click, the element is automatically moved to the centre of the screen (*automatic zoom*). Zooming is also possible by double-click or by using the slider to the left of the canvas. Like Denim and Damask, we offer a slider for zooming in and out (see Figure 4, 8). In Figure 4, the slider offers two choices. More options are only available when the user has chosen to zoom either into the page view or into the model view (see Figure 3). As scenario and page flow levels are always available on the slider, it also serves as a history of zoom-operations and enables us to abstain from other means of navigation such as a history layer [15].

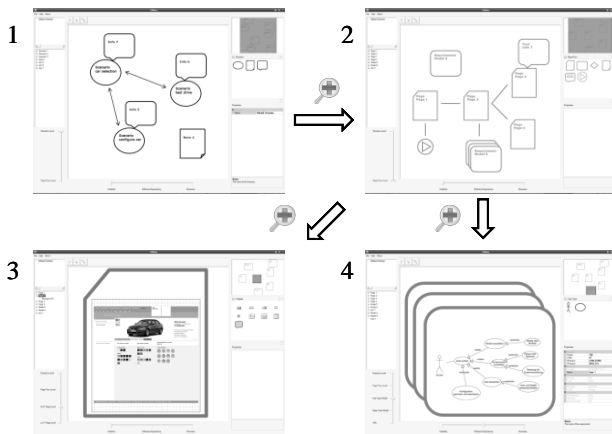


Figure 6: Zooming into pages and models

Through a semantic drill-down into a scenario object (see Figure 6, 1), the underlying *page flow* becomes visible (see Figure 6, 2 and Figure 7). The semantic zoom operation corresponds to an upwards-vertical navigation in the hierarchy of levels of abstraction. Consequently, the user now sees the overview on all pages that are contained in the corresponding usage scenario (see Figure 7).

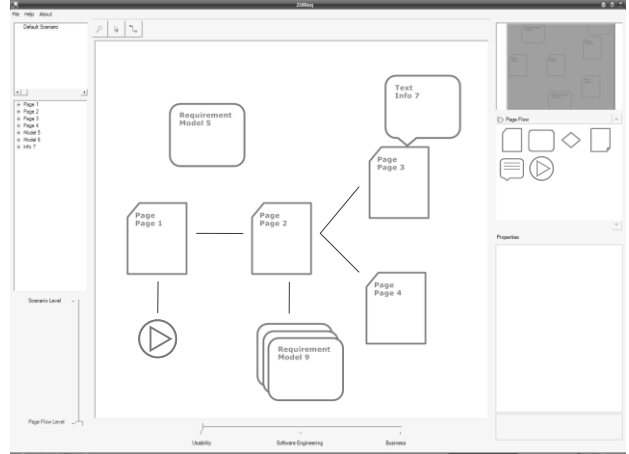


Figure 7: Page flow level of a scenario

The iconified representations of pages can be linked in terms of simple, single transitions or dependent on dialogue sequence and user decisions. They can be freely arranged on the screen and one can be set as the starting page. Changes - for instance insert, delete and exchange pages - are possible at any time. From the *page flow layer* (see Figure 6, 1), the stakeholder can then either access the UI of a single page (see Figure 6, 3) in detail or zoom into the graphical models (see Figure 6, 4) that they are part of (e.g. a Task Model).

Following our example, e.g. a business stakeholder can now add more pages or specify detailed requirements for the page flow at hand. On the one hand, the user can add information objects to add discipline-specific textual description (i.e. balloon objects, see Figure 5). On the other hand, the user can add sophisticated requirements models. Following our outlined hierarchy of modelling languages, these can be text-based ones in the first instance (see Figure 8) and graphical notations at later stages.

Requirements can be attached either to single pages or to the scenario as a whole. By zooming in to a requirements object on the canvas, its representation is dependent on the current user role (see Figure 8). The stakeholder views are also distinguished by using colour coding. Accordingly, each identified group of stakeholders gets its own specific toolbox for the modelling of requirements.



Figure 8: Detailed view of a textual business requirement

If it is necessary for understanding the dependencies to other stakeholder roles and responsibilities, a user can now change his view on the requirements by using the slider. According to the selection, the toolbox is adapted and shows the means of expression of the selected discipline.

It is also possible to create different versions of a model. This allows the tracing of changes to requirements. The different versions are arranged behind one another on the main canvas (see Figure 9, lower screen). The user can browse through the versions by selecting a particular panel or by running a slide show.

To create the different models, the user is offered modelling objects in the toolbox, these objects again being adapted to the current user role. For example, a usability expert could add user role models, task models or use UIML to describe the UI of the software before he decides to build the UI.

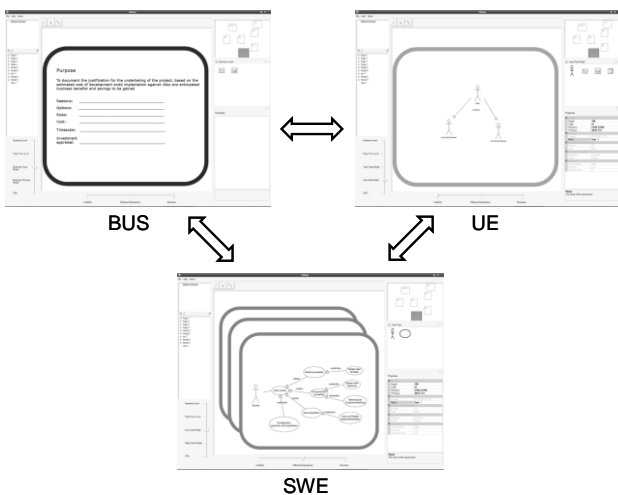


Figure 9: Discipline-specific modelling with different levels of abstraction

Finally, the *page view*, which is accessed through the *page flow level*, allows the user to create and modify the UI and hence the look and feel of the visual specification. In order to develop good usability, all stakeholders have to merge their requirements with the interactive simulation at the *interaction layer* (see Figure 2). Usually, the interaction will be designed after stakeholders have been able to agree upon requirements and understood the interdependencies between their models. As

with scenarios, prototypes are well known across disciplines [19].



Figure 10: Low- (left) and hi-fi (right) page views

Apart from a discrimination between low- and high-fidelity prototypes (see Figure 10), we therefore do not further distinguish between discipline-specific prototyping methods at this level. Instead, we only fade in annotations (e.g. made by reviewers) according to stakeholder expertise. Such notes enable the stakeholders to e.g. write down change requests [18].

Placeholders and symbols can be used to create a low-fidelity page. The main focus then is on segmentation and arranging screen elements and regions. Pictures and graphics can be hand drawn with a pencil tool (which is similar to *Denim* and *Damask*). Similar to building models, pages of low- and high-fidelity could exist in different versions. Versions can e.g. visualize distinct designs at each iteration. Once abstract pages are developed according to the requirements, a high-fidelity representation can be created (see Figure 11) to model interactive UI behavior. For detailed prototyping, the user is offered standard widgets and can include images as well as other embedded objects. Similar to drawing PowerPoint slides, this works without coding. With regards to the needs of software engineers, we do nevertheless envisage integrating the possibility for scripting in later versions of our tool. Hence, the corresponding user role would have the chance to create even more interactive content (e.g. based on the work of the usability engineer).

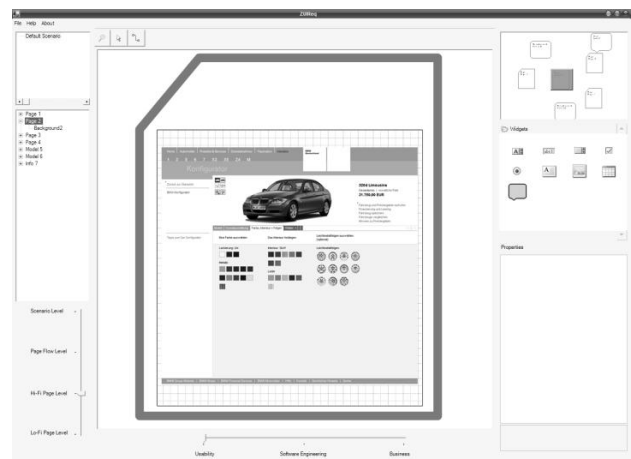


Figure 11: High-fidelity page design

Both low- and high-fidelity prototyping can be supported by creating templates and master sheets. This is very helpful for the rapid prototyping and generic changes of similar pages. Although we have not yet implemented a recommendation wizard for UI designers, we are inspired by the UI patterns that are available in *Damask*.

During prototyping, stakeholders can keep track of what has to be designed in detail by switching between the different representations. Furthermore, they can at any stage cross-check the UI design with the underlying requirements models by changing to the model levels (see Figure 3). Therefore, the user

could e.g. use the jump zoom operation with the help of the tree-view component.

Once the specification process is finished, all material can be saved in a XML file. This allows the forwarding of the specification to the contractor in a machine-readable format. Furthermore, the description of models and UI design allows the generation of running simulations [11]. In the following, we will describe the technical architecture of our tool and indicate the current status of our software construction site.

6. TECHNICAL IMPLEMENTATION

In order to implement our concepts, we used modern utilities and programming languages. We built our application in C# with the help of Visual Studio and the .NET Framework from Microsoft. For addressing the various problems of a complex specification tool, we used the famous model-view-controller (MVC) pattern (see Figure 12). Hence, the visual representation and data management are separated from the interaction of the user with both representations. The controller layer is used to control and synchronize views with their data representation.

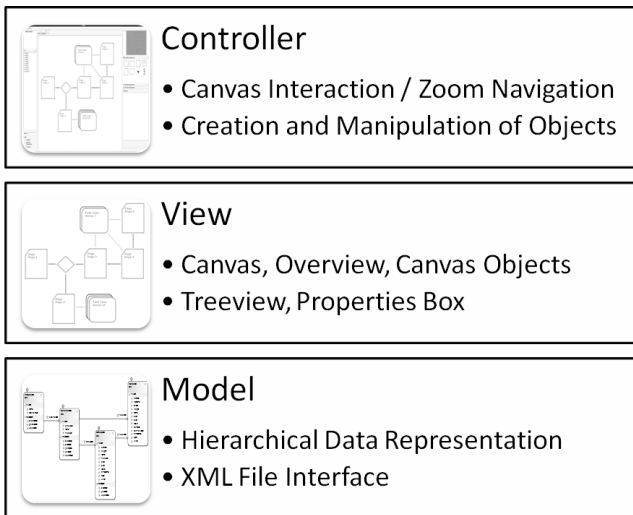


Figure 12: Model-view-controller pattern

Both ZUI canvas and overview window were implemented utilizing the Piccolo framework [3]. Piccolo is a high-level API for creating and interacting with two-dimensional structures based on the scene graph model. The .NET version of Piccolo is based on the underlying GDI+ layer of the .NET Framework for rendering two-dimensional graphics. Piccolo offers a hierarchical structure of objects and cameras, which allows the developer to create, group, and manipulate objects in a simple manner. Piccolo's infrastructure also provides a well-designed rendering and interaction management. For the representation of requirements objects, canvas classes from Piccolo were extended and customized accordingly. For the design and rendering of graphical objects on the canvas surface, the Enhanced Metafile Format (EMF) is used; this is natively supported by GDI+ and therefore has a good rendering performance. The EMF format supports both vector graphics and bitmaps. It is therefore suitable for all kinds of diagrams or pictures needed to represent abstract models. Other graphical elements such as embedded PDF files, documents or other control elements are already supported by Piccolo.

The interaction of the user with views and the data representation is synchronized concurrently within the Controller layer. Navigation within the ZUI and the selection of objects are implemented by an extended navigation handler.

Creation and manipulation of objects on the canvas and the internal representation are handled by a single interface. By using this design it is guaranteed that the two representations remain consistent at any one time. When changes occur, e.g. initiated by user interaction to one of the different views, the other views are updated immediately. Views are synchronized by employing the Observer Pattern. All views register themselves to a global observing object that provides instant update messages for all views.

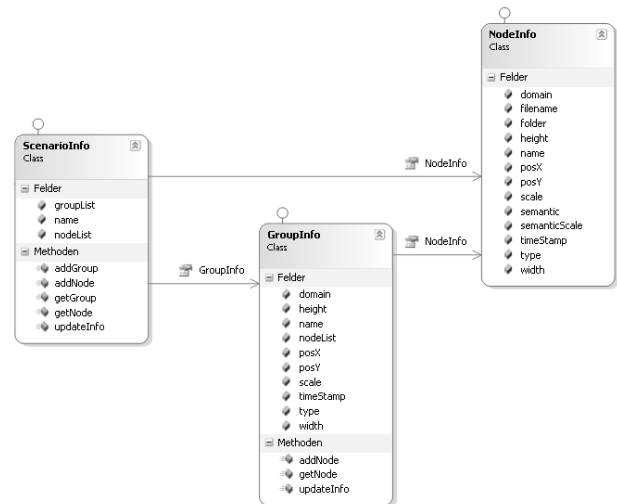


Figure 13: Modelling objects and their relations in UML (simplified outline)

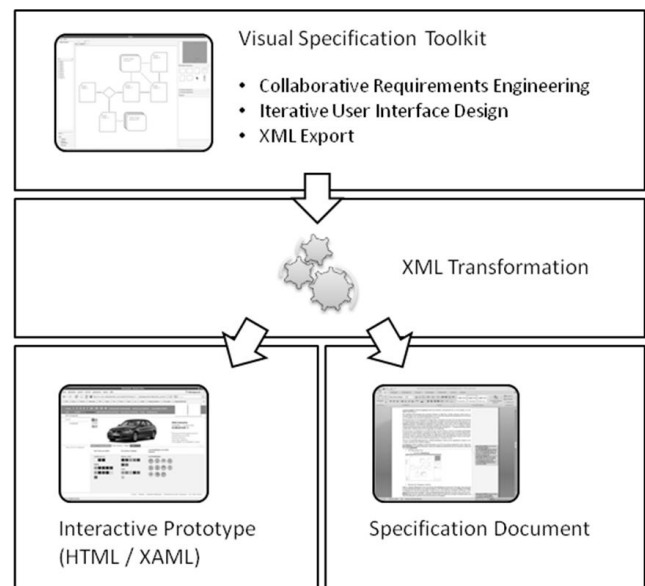


Figure 14: Generation of prototype and documentation with code generator

Figure 13 shows an outline of the relationship of drawing (i.e. modelling) objects in our implementation. The UML diagram shows that each scenario can consist of both group and node items. Node items are single objects such as shapes that are typically offered in the drawing toolbox for creating models on the canvas. Group items are objects that summarize a collection of related nodes. For example, the shape that represents a scenario is a placeholder for the set of objects it is composed of, which can be accessed by zooming into the subsequent information layer.

Together with the entire data model, the drawing objects and their properties can be exported into a XML file. All instances of objects can be serialized to a file output stream.

Based on our experience with model-based development, we know that we can use XML to automatically create a system simulation (and supporting material such as documentation) at a later stage (see Figure 14). As we have already implemented this in a different project [11], we have not yet focused our efforts on this part of our tool.

7. SUMMARY & CONCLUSION

We began to build an UI specification tool that utilizes various techniques of information visualization and human-computer interaction. By integrating overview & detail and zooming, we implemented a means for supporting requirements specification processes with multiple stakeholders. In contrast to many existing (CASE) tools, we try to bridge between the disciplines by providing related modelling languages rather than inventing new terminologies. With regards to collaborative design, we focus on ease of use and ease of understanding.

Our research is related to Denim and Damask. These tools are based on a *paper and pencil* prototyping approach. They enable to sketch out abstract UIs and create page-flows easily. We extended their idea by having an eye on more disciplines and by being more formal, model-based and focused on reusability.

As our tool is targeted towards prototyping and visual UI specification, we do not expect the stakeholders to use our approach to create the final UI, nor do we expect the generated UI to be used without modification. Our goal is to increase the opportunities for developing high-quality systems. Thanks to related research, we concentrate on implementing the means of visualization and interaction and place less emphasis on technical issues such as the ZOOM approach [17] or code generation [11].

The specification tool that we present here is a prototype and still being developed at our human-computer interaction lab. We will continue with enhancing its functionality, as some features are only implemented prototypically. As our system is not yet ready for application under project conditions, we could not evaluate its actual contribution³. But based on our experience and following the results of our survey of current development practice, we believe our approach to be noteworthy and instrumental in engineering interactive systems.

8. REFERENCES

- [1] Atlantic Systems Guild Limited (Volere), Requirements Tools: <http://www.volere.co.uk/tools.htm>
- [2] Bederson, B. B.; Boltman, A. Does Animation Help Users Build Mental Maps of Spatial Information? Tech Report CS-TR-3964, Computer Science Department, University of Maryland, College Park, MD, 1998
- [3] Bederson, B. B., Grosjean, J., & Meyer, J. Toolkit Design for Interactive Structured Graphics. *IEEE Transactions on Software Engineering*, 30 (8), 2004, 535-546.
- [4] Bock, C., Zühlke, D. Model-driven HMI development – Can Meta-CASE tools relieve the pain? In *Proceedings of the First International Workshop on Metamodeling – Utilization in Software Engineering (MUSE)*, Setúbal, Portugal, September 11, 2006, 312-319.
- [5] Bruening, J., Dittmar, A., Forbrig, P., Reichart, D. Getting SW Engineers on board: Task Modelling with Activity Diagrams. In *Proceedings of the Engineering Interactive Systems Conference (EIS 2007)*, Salamanca, Spain, 2007
- [6] Buering, T., Gerken, J., Reiterer, H. Usability of Overview-Supported Zooming on Small Screens with Regard to Individual Differences in Spatial Ability. In *Proceedings of the Advanced visual interfaces (AVI) 2006*, ACM Press, 2006
- [7] Furnas, G. W., Zhang, X. MuSE: A multiscale editor. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology (UIST '98, San Francisco, Calif., Nov. 1-4)*. E. Mynatt and R. Jacob, Eds. ACM Press, New York, N.Y., 1998, 107-116.
- [8] Gundelsweiler, F., Memmel, T., Reiterer, H. ZEUS: Zoomable Explorative User Interface for Searching and Object Presentation, In *Proceedings of the HCI International 2007*, Beijing, China, July 22nd-27th, 2007
- [9] Hoffmann M., Kühn, N., Weber, M., Bittner, M. Requirements for Requirements Management Tools, Pin proceedings of the 13th IEEE International Conference on Requirements Engineering, 2004, Berlin, Germany, 2004, 301-308.
- [10] Lin, J., Landay, James A. Damask: A Tool for Early-Stage Design and Prototyping of Multi-Device User Interfaces. In *Proceedings of The 8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing)*, San Francisco, CA, September 26-28, 2002, 573-580
- [11] Memmel T., Bock, C., Reiterer, H. Model-driven prototyping for corporate software specification, In *Proceedings of the Engineering Interactive Systems Conference (EIS 2007)*, Salamanca, Spain, 2007
- [12] Memmel T., Gundelsweiler, F., Reiterer, H. Prototyping Corporate User Interfaces – Towards A Visual Specification Of Interactive Systems. In *Proceedings of the IASTED-HCI 2007*, Chamonix, France, 2007
- [13] Newman, M. W., Jason, J. L., Hong, I., Landay, J. A. DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. In *Human-Computer Interaction*, 2003. 18(3): 259-324.
- [14] Perlin, K., Fox, D. Pad: An alternative approach to the computer interface. In *Proceedings of the 20th Annual ACM Conference on Computer Graphics (SIGGRAPH '93, Anaheim, Calif., Aug. 2-6)*. J. T. Kajiya, Ed. ACM Press, New York, N.Y., 1993, 57-64.
- [15] Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. Context and Interaction in Zoomable User Interfaces. In *Proceedings of the AVI 2000 Conference*, 2000, 227-231
- [16] Potts, C., Takahashi, K., Anton, A.I. Inquiry Based Requirements Analysis. *IEEE Software*, Volume 11, Issue 2, 1994, 21-32.
- [17] Qin, L., Liu, H., Jones, C., Jia, X. An Integrated Event-Driven Framework Supporting MDD. In *Proceedings of the Midwest Software Engineering Conference (MSEC'04)*, April 30th 2004 Chicago, IL, USA, 2004, 32-44

³ Experience from applying our tool in the field as well as a stable version of the demonstrator will be available at the conference.

- [18] Rashid, A., Meder, D., Wiesenberger, J., Behm, A. Visual Requirement Specification In End-User Participation. In *First International Workshop on Multimedia Requirements Engineering - Beyond Mere Descriptions (MeRE'06) on 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis/St. Paul, Minnesota, USA, September 11-15, 2006.
- [19] Seffah, A., Desmarais, M. C., and Metzker E. HCI, usability and software engineering integration: present and future. In: Seffah, A., Gulliksen, J., Desmarais, M. C. (eds.), *Human-centered software engineering – integrating usability in the development process*, Springer, Dordrecht, Netherlands, 2005, 35-57
- [20] Standish Group, *Chaos Reports 2003*: <http://www.standishgroup.com>.
- [21] Ware, C. *Information Visualization: Perception for Design*. Morgan Kaufmann, San Fransisco, California, 2004
- [22] Zave, P., Jackson, M. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*. Volume 6, Issue 1, 1997, pp. 1-30