Knowledge System Prototyping for Usability Engineering

Martina Freiberg, Johannes Mitlmeier, Joachim Baumeister, Frank Puppe
University of Würzburg
D-97074, Würzburg, Germany
freiberg/joba/puppe@informatik.uni-wuerzburg.de

Abstract

Knowledge-based consultation and documentation systems are widely distributed in industrial and medical environments today. Yet, their implementation still is a tedious and costly task. Furthermore, the aspect of usability—which is in principle of critical importance for those systems—is often nearly unconsidered. We argue, that tailored UI prototyping can help to tackle both issues. Therefore, we propose a UI prototyping tool for knowledge-based systems, intended to enhance knowledge systems engineering in itself, and to foster usability engineering in that context.

Keywords: Knowledge-based System, User Interface Prototyping, Usability, Human Computer Interaction

1 Introduction

Knowledge systems (KS)—in the context of our paper knowledge-based consultation and documentation systems—are applied in various industrial and medical environments today. Regarding their development, it has to be differentiated between knowledge system engineering (KS Engineering) and knowledge engineering (KE). The former comprises the entire development process of a knowledge-based system, including especially its UI and interaction design; the latter specifically addresses the definition and formalization of the required knowledge, e.g., the terminology, or explicit problem-solving knowledge.

KS Engineering Pitfalls and SE Solutions

Despite the widespread use of knowledge systems, and consequently increasing research efforts regarding their development in the last decades, KS Engineering still remains a tedious and complex task. Among the main pitfalls are high development costs, both in terms of money and time. Thereby, the sub-task KE alone often causes a major part of the expenses, hence influencing other KS Engineering activities:

In many cases, UI- and interaction design in general—or a more targeted comparison of several equal design options—would require more attention. For knowledge based systems, it is further of critical importance, that they are intuitive and easy to use as to not distract the users from the often difficult, domain-specific jobs, they are intended to support (e.g., decision-support in medical contexts). Yet, despite various recognized general usability engineering approaches, usability issues often also remain almost unconsidered. Finally, the often still missing true understanding of such systems and their benefits on the side of potential customers, can make it difficult to promote respective projects in the first place due to the overall complexity and costs.

In general software engineering (SE) and in human-computer interaction (HCI), user interface (UI) prototyping is an established method for iterative specification and refinement before implementing the productive system [3; 4]. The increased flexibility arising from prototyping-based specification and design offers the chance to adapt system (and especially interface) requirements to changing base requirements or customer wishes in a more efficient, inexpensive manner. The affordability of the approach also permits the early evaluation and comparison of design alternatives. In providing a (visual) basis for communication, UI prototyping can help to specify requirements more precisely. Thus, the potential risks of fundamental misunderstandings, and a resulting, more expensive redesign of central conceptions at a later stage of the project, can be reduced. Also, the overall system vision can be communicated and refined more easily with the help of an appropriate prototype.

In tailoring UI prototyping for knowledge systems, we aim at exploiting those advantages. Particularly, we intend to foster affordable, pragmatic KS Engineering, that both helps to promote respective projects in the first place, and alleviates the overall task. Our approach intentionally focusses on interface and interaction design of KS, and on an increased integration of usability considerations in the process. Regarding the specification and formalization of the required terminology and explicit knowledge of a KS, there exist various established KE methods today, each of which can be equally well applied—thus, we do not further discuss or value their suitability here.

Related Work

In general SE and in HCI there exist numerous classifying approaches regarding prototyping and the prototyping process, e.g., see [3; 4; 5; 8; 11]. Apart from manifold general prototyping tools and methodologies available—see Beaudouin-Lafon and Mackay [4] for an overview—also tailored tools have been developed in various specific domains; examples are the field of multimodal interaction research [12], or the field of cross device interface design [10]. To the best of our knowledge, no prototyping approaches or tools exist, that specifically address knowledge systems according to our definition (see Section 3.1).

Lim et al. [9] note that in HCI, prototypes to date

mainly are used for evaluation purposes—such as usability testing—and that in SE, prototyping mostly constitutes a means for supporting requirements engineering; in extension to that, they suggest prototypes as tools for designers to frame, refine, and discover options in a design space. Regarding an integration of software- and usability engineering, Memmel et al. [13] similarly claim an incorporation of visual requirements engineering—based on appropriate prototypes—much earlier in the overall process. Merging those insights, we propose UI prototyping as a rather pragmatic means for KS requirements engineering. As opposed to the above approaches, we explicitly address knowledge systems, that exhibit some specific characteristics. First, KS mostly consist of a rather fixed set of UI elements and user-system interactions; most often, for example, questions are presented, answered by the user, and cause a certain follow-up system (re)action. An adequate prototyping tool thus firstly should support the design of such elementary elements as flexibly as possible. For realistically emulating actual knowledge systems, additionally the imitation—or actual integration—of the underlying explicit knowledge needs to be supported, as to enable a reasonable judgement regarding the applicability and usability of the overall future system.

When designing with usability in mind, some kind of iterative process is highly advisable [14]. Angele et al. [1] introduce a cyclic process model for developing knowledge-based systems; it incorporates prototyping techniques, but furthermore also formal specification and KE activities, thus constituting an entire, rather heavyweight engineering approach. Contrastingly, we suggest an extension of the rather lightweight *Agile Process Model* [2]; thereby, the focus is on providing an overall pragmatic method of KS Engineering and on enabling a rather inexpensive integration of usability activities.

In summary, we contribute to current research by

- proposing an overall approach that integrates efficient, affordable KS Engineering and usability engineering.
- introducing the UI prototyping and engineering tool *ProET*, specifically tailored for the design of knowledge-based systems

The remainder of the paper is organized as follows: In Section 2, we discuss a customized, prototyping-based KS Engineering process, as well as its potential regarding an integration of usability activities. We present *ProET*, an UI prototyping and engineering tool for knowledge-based systems, in Section 3. In Section 4, we report on experiences of exemplarily recreating existing knowledge systems with the tool, and on its consequential current scope and limitations. We conclude with a summary and an outlook to further research directions in Section 5.

2 Pragmatic KS Engineering for Usability

Regarding knowledge system development and knowledge engineering, there exist diverse approaches today, such as *CommonKADS*, *MIKE*, or adaptions of the classical *stage-based* and *incremental* software development models. Yet, for the success of knowledge system projects also and especially regarding small to mid-sized companies, a pragmatic approach—affordable and efficient regarding time and effort—is essential, c.f. [2]. Especially for promoting such projects in the first place, it is important to quickly

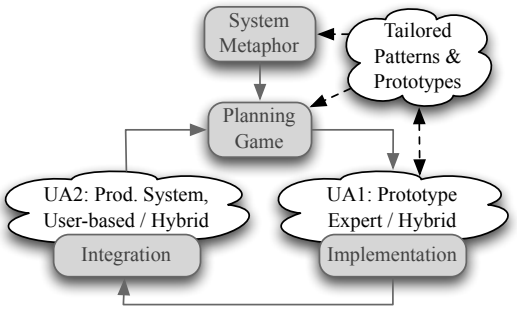

Figure 1: Extended Agile Process Model.

come up with first solutions, e.g., in the form of prototypes or example implementations. In this respect, we made positive experiences with applying the *Agile Process Model*, described in [2]. However, that model emphasizes knowledge base development, not yet taking much into account the design of the target system's interface, or usability traits.

Targeting an overall approach that supports pragmatic, affordable, and usability-involving KS Engineering, we propose the extension of the *Agile Process Model* by integrating pattern-based design, prototyping, and usability techniques into the original model. Figure 1 introduces the entire resulting *Extended Agile Process Model*. Although pattern integration and respective activities are included in the following for reasons of completeness, their more detailed discussion is part of further work, see [7].

The gray parts of Figure 1 represent the original model, consisting of the four phases *System Metaphor*, *Planning Game*, *Implementation*, and *Integration*. For a detailed discussion, see [2].

Basically, tailored patterns and prototyping can support both *System Metaphor* and *Planning Game*. In *System Metaphor*, the system objectives are defined by developers and customers. Based on appropriate patterns and corresponding implementation examples, a basic idea can be developed more easily. Thereby, patterns can be assessed either manually, or by using a tailored recommender system, that suggests patterns matching the target context. Prototypes, that also provide the relevant user-system interactions, further support that process by presenting a realistic simulation of a potentially resulting system as opposed to the static, visual depiction of knowledge system examples provided by the patterns.

The *Planning Game* defines the scope and prioritization of development tasks. Here, patterns ease the analysis and valuation of system requirements—taking place during the *Exploration* sub-phase of the planning game—by providing clear specifications of required features and interactions. Additionally, prototyping supports that task by allowing for actually trying out (and thus better evaluating) relevant functionalities.

With regards to *Usability Activities*, the original model can be extended both regarding *Implementation* and *Integration* (Figure 1, UA1, UA2). The basic model defines *Implementation* as a test-first activity—i.e., before actually implementing new or additional features, the corresponding tests for assuring their correctness are developed. This can be expanded by an evaluation-first activity, in the sense that based on the formerly created pro-

totypes, usability issues are assessed and valued first, before continuing with test-first implementation as defined by the model. Performing prototype-based usability evaluation offers the chance to reveal defects of the design at early stages. This can considerably lower development costs, as the adaption/revision of a prototype is rather inexpensive, in contrast to adapting an preliminarily implemented, or even already productive system. Without going into detail here, at that stage, expert- or hybrid approaches (according to a categorization suggested in [6]) seem to be most appropriate. For example, rather light-weight techniques—as feature-/consistency inspection—but also more comprehensive methods—as heuristic evaluation or expert walkthrough—are performed by the developer ("expert"). In case even future users—e.g., project partners or their employees—are available, hybrid methods such as pluralistic walkthrough or participatory/cooperative heuristic evaluation potentially can provide the most benefits. However, some of those techniques require at least a partly functional system—as explained in Section 3, also fundamental interactions of knowledge systems can be designed/simulated with the suggested prototyping tool *ProET*. Thus, those techniques are (at least partly) applicable to the developed UI prototypes.

During *Integration*, the implemented functionality is added to the productive system, using integration tests for assuring its overall correctness and integrity. Such testing can be extended by usability evaluation activities that check, whether the system still meets the specified usability goals. As *Integration* results in a running version of the productive system, it is not only possible, but rather highly advisable, to evaluate the applicability of the system in the target context with real users. Thus, not only hybrid, but also purely user-based usability evaluation techniques are beneficial—example techniques are querying, user studies, or controlled experiments. Additionally, again also *Hybrid Approaches* may also provide valuable insights regarding the actual use of the knowledge system and potential, remaining defects.

The suggested approach aims at turning overall KS Engineering into a more pragmatic process, equally suitable for promoting KS projects—by quickly setting up and presenting actual KS examples (prototypes) to customers—and for specifying requirements as well as the system design in a more agile manner. Due to the highly iterative process, that also involves usability evaluation activities at specified stages, potential system flaws may be detected, or even prevented more effectively.

3 The Prototyping Tool *ProET*

In this section, we introduce the prototyping and engineering tool *ProET*, that we are developing to support affordable and efficient KS Engineering, as well as to foster an eased integration of usability-related activities in the overall process. Therefore, we first define the specific type of target systems, as well as typical components those systems are built of. Afterwards, we introduce *ProET* and its workflow of creating prototypes in more detail.

3.1 Target Knowledge Systems and Components

By *knowledge system*, we understand systems that may implement various forms of knowledge—such as rules, or covering models—to support the user as efficiently as possible in performing the task at hand. Thereby, we specifically think of consultation and documentation tasks—in the first case, the system provides decision-support or recommendations regarding a specified problem area (e.g., in medical or fault detection contexts); in the second case, users are assisted in entering a certain set of data and the system ensures its quality (e.g., measured by completeness and correctness). The initial capabilities of the tool are based on our past experiences with developing knowledge-based systems. For the greater part, those were implemented as *web-based systems*—not necessarily meaning they are made available to large masses of users via internet, but in the general sense of "running in a browser". Thus, the tool specifically supports web-based consultation and documentation systems engineering.

For those target systems, typically a certain set of *visible knowledge components* can be identified:
- *Questions:* Requesting required input data from the user
- *Questionnaires:* May be used to group the (potentially large set of) questions
- *Answers:* A fixed set of reasonable input data (answer alternatives) to choose from, or free text input facilities
- *Solutions:* Fault or medical diagnoses, or action recommendations, that are derived by the included diagnosis knowledge (*invisible knowledge components*, e.g., rules)
- *Ancillary Information:* Informal knowledge representations, detailed elaborations of questions/solutions, or add-on information regarding the overall consultation/documentation progress.

Those components form the conceptual basis of the widgets currently supported by *ProET*. Thus, they constitute the elementary items that are to be specified in the declarative prototype specification file (as described in the following section in more detail).

3.2 Introducing *ProET*

The **pro**totyping and **e**ngineering **t**ool *ProET* is an UI prototyping tool specifically tailored for web-based consultation and documentation systems. Thereby, prototyping is supported gradually: First, exemplary system definitions (and corresponding templates/styles) allow for quickly and easily creating and exploring the basic collection of knowledge systems supported so far. Based on those available specifications, adapted KS prototypes can be created in a copy & modify manner, where the degree of modification can vary arbitrarily. With the extensibility of the tool, finally also entirely different interfaces/UI components can be developed and integrated, if required (see Section "Extending *ProET*").

The tool supports two basic modes of prototyping: Complete specification of all textual elements, as well as their (partly or entire) auto-generation.

The first variant is useful for prototyping and evaluating concrete KS ideas. The required visible knowledge components (such as questions, answers...) are defined in the declarative prototype specification file. As also elementary interactions—e.g. coloring the next suggested question, or hiding/unfolding parts of the dialog—are available, future knowledge systems can be simulated rather realistically. Thus, it can be examined whether a chosen UI/interaction design is suitable in a given, domain-specific context.

The option of auto-generating textual elements, further-

more allows for a more design-oriented prototyping: Not having to consider the specification of actually reasonable knowledge base elements simplifies the concentration on UI/interaction design questions. This can be helpful in case several designs are to be compared against each other, or provided that general design issues need to be evaluated, independent from any future system.

Technical Basis

ProET is a JAVA application that integrates several web-based technologies for engineering UI prototypes of web-based knowledge systems. The created prototype is HTML-based, enriched by JavaScript/AJAX for interactivity and styled by CSS. The tool is probably most comfortably used from within some kind of IDE—such as Eclipse[1]—that supports editing of the required file formats, as well as an easy management of the project itself.

Prototyping Workflow

To provide a first impression, Figure 2 presents a questionary-style, partly auto-generated consultation system prototype. Figure 2 (A) displays a page containing concretely specified questions and answers; another page of the same prototype, with auto-generated textual elements, is shown in Figure 2 (B). Prototyping with ProET currently is purely text-based. We use the above prototype as a running example when introducing the three types of specification files required for prototyping with *ProET*:

- An *XML-based* specification file (central prototype- and textual content specification)
- *String Template* files (creating HTML-/JavaScript-based fragments for each defined component)
- *CSS* files (design definition of specified components)

Once the specification via these files is finished, the prototype is assembled: Filling in the textual contents from the XML specification, HTML-/JavaScript-based component representations are created using the String Templates for the framework and CSS for the concrete styling.

XML-based, Elementary Prototype Specification The elementary prototype specification—i.e., its skeletal structure, consisting of the textual elements as well as of their basic UI properties (e.g., whether the question-style is one-choice or multiple-choice)—is provided in an XML-based format. For each of the visible (textual) knowledge system components, matching tags are provided—e.g., an `<answer>` tag is used for defining answers. Within those tags, the fundamental UI properties are specified as attributes—a multiple choice question, for example, is defined using `<question answer-type='mc'>`. Furthermore, the XML specification references the respective template- and CSS files, that are additionally required for creating the prototype.

Figure 2 (C) presents an excerpt of the file used for specifying the prototype shown in (A). Excluding the standard XML-header, the framing `<dialog...>` tag is the topmost element; there, the knowledge system type—here: `type='gen'`, referring to a predefined, (partly) auto-generated prototype style—is defined, as well as references to the respective template namespace, and style files. Figure 2 (D) exemplifies the specification of a questionnaire (here still named `page`) that consists of several questions; due to space reasons, only the detailed definition of the first question is printed completely. The example illustrates the specification of the question `Do you like surveys?`, the setting of its basic UI style by `answer-type='oc'`, as well as of its three answer alternatives `Yes, No, Neither...Nor`. Furthermore, the definition of generated textual elements is exemplified in Figure 2 (E). This is achieved by using `<generate>` tags, that can be attributed by the desired number of questions of the generated page (`num-questions='3'`) and respective answer alternatives (`num-answers='3'`), by the text lengths (`question-/answer-length='...`) or by the basic answer-style (as above). Regarding the auto-generation, it is possible to either define the number of questions (answers, text/answer length) strictly—e.g. `num-questions='3'`—or to specify a more flexible range—e.g. `num-questions='2-5'`—resulting in randomly calculated minimum of two and maximum of five questions.

String Templates String Template files provide the HTML-counterparts for each of the defined knowledge system components. When creating a prototype, one framing template is defined for the dialog as a whole—Figure 2 (F)—which contains the skeletal HTML-framework. From there, sub-templates are referenced, that define the HTML fragment of the respective components separately—in the figure, `$children$` means that the also depicted template for the *children* of the *dialog-content* element—which are pages—is inserted at this point. Splitting the UI templates into framing- and sub-templates, according to the corresponding component definitions, provides the advantage that the separate component templates, e.g. for a page, can be reused in several different prototypes.

CSS for UI styling The actual styling/design of the components is finally specified using standard *CSS*. Most basically, each knowledge system component can be globally styled by a CSS class with a matching name—e.g., common properties of all questions can be set by a `.question` class. Yet, a more fine-granular styling is also possible—additional, element-specific CSS classes or IDs can be set within the template files, and then can also be specified separately in the CSS file.

Interactivity With *ProET* not only static UI designs can be created, but also elementary interactivity that would be expected of an actually implemented knowledge system. Examples are the interactive coloring of questions (or solutions) according to their current status—e.g., answered/suggested next (or established/excluded)—or hiding parts of the questions until another defined question was answered by the user. Such interactive behavior is achieved by the usage of JavaScript in *ProET*. The required functionality thereby is defined in separate JS files, whereas the corresponding function calls are inserted in the String Template file where needed—e.g., if a given element should provide some interactivity on mouse-click, in the template file like `onClick="javascript:doSomething()` is inserted within the respective tag defining that element.

[1] http://www.eclipse.org/

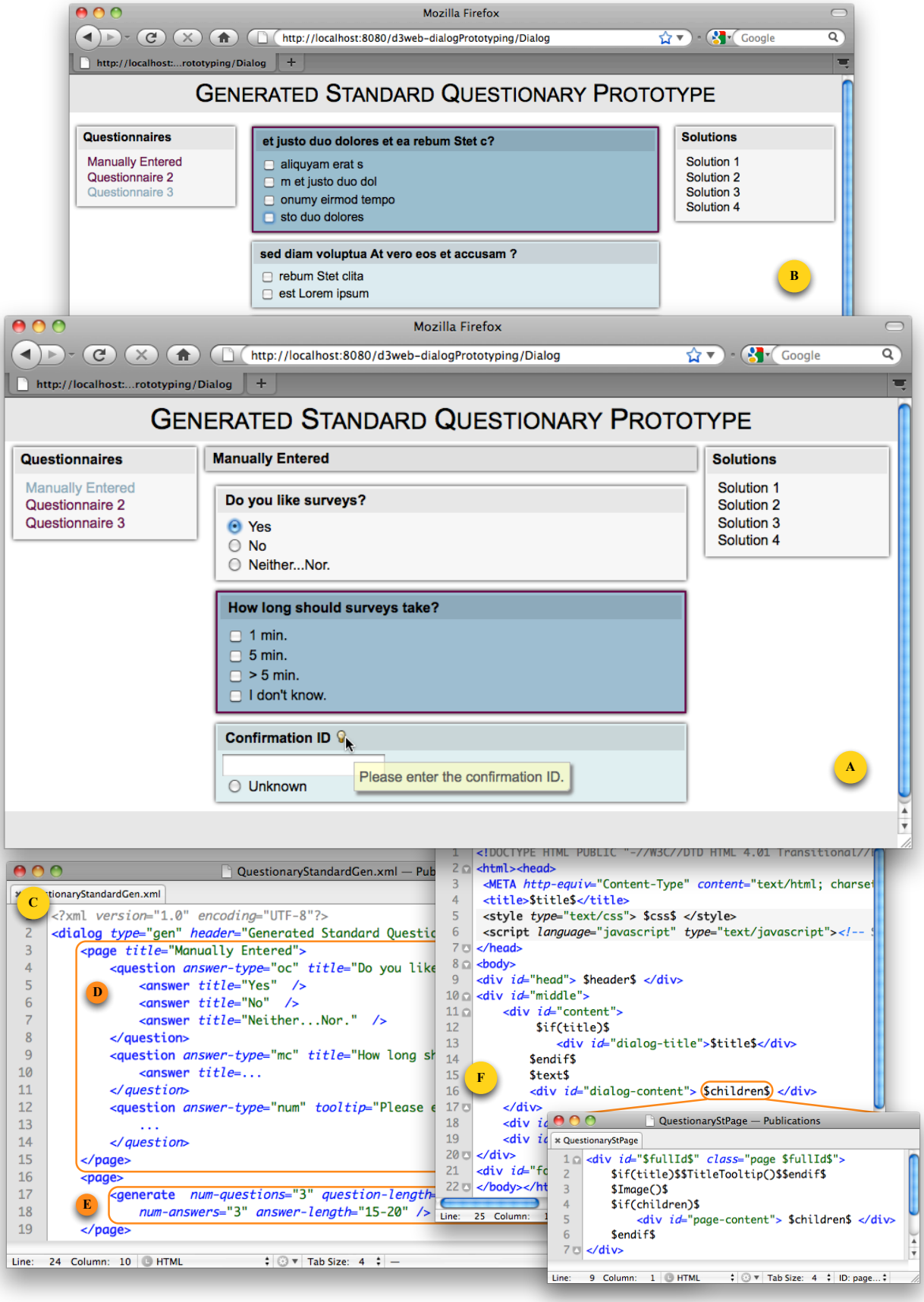

Figure 2: Partly auto-generated prototype of a questionary-like consultation system—manually entered questions (A), auto-generated questions (B), the corresponding prototype specification (C), and an exemplary template file (F).

Extending *ProET*

Apart from just adapting existing system templates/designs, it is also possible to extend the tool by defining entirely new elements. Thereby it has to be differentiated between rather simple extensions—such as new, XML-based prototype specifications—and more sophisticated extensions regarding entirely novel components—for example, progress indicators or the like as separate elements.

Regarding the definition of a new prototype, simply a new XML specification is created, using the available set of tags and corresponding attributes. New/adapted String Templates for components can be introduced by providing a corresponding `type` attribute within the dialog tag and a corresponding String Template file; the `type`−value then is matched by a certain mechanism with potential templates until the most appropriate one is found; due to space reasons, the entire naming-and-matching mechanism is not explained in detail here. Finally, to modify not the basic form of prototype but only its UI design, additional CSS files may be created and just referenced in the `<dialog...>` tag of the specification. This permits an easy exchange and comparison of several design options.

The extension of the tool by entirely new components, on the other hand, also is possible, yet more complex. This additionally requires an adaption of the underlying Java code (e.g., create new container classes for the element, adapt the XML parsers to correctly parse that new elements, and so on). The details of this entire extension process, however, would fall out of the scope of this paper.

4 Experiences with ProET

For a preliminary assessment of the capabilities of *ProET*, we recreated knowledge systems that have been developed by our department in the past. Thereby, the first goal was to match those original systems as closely as possible. Figures 3 and 4 present the outcome of replicating two rather different systems; each figure shows both the original system in the background (A) and the prototype in the foreground (B). In the following, we first shortly introduce each system and summarize the insights regarding its replication with *ProET*. Based on that, we discuss the scope and limitations of the tool in the subsequent section.

4.1 Knowledge System Replication

The *Consultation on Rheumatic Disease*, Figure 3, served as the first case study. Based on the entered symptoms, that system consults the user as to whether a rheumatic disease is probable. Basically, a questionary-style is implemented, meaning that the system presents more than one question at a time to the user. Thereby, a certain coloring metaphor is applied for supporting an answering of the questions in the most reasonable sequence: Questions, that are already answered are colored gray; not yet answered questions are colored yellow, and the suggested next question is colored green.

This coloring-based interactivity is also mirrored in the prototype. There, JavaScript-based techniques are used to change the coloring according to the user's actions. Thus, the basic system interaction and styling can be realized by the means of the prototyping tool quite well. In some minor aspects, however, the prototype differs from the original system. First, a seemingly more loosely assembled interface appearance; whereas in the original system kind of a (visible) table-based layout was used, the prototype builds on a CSS-based layout, defining questions as separate elements and rendering them without using tables at all. If desired, however, it is with some effort possible, to adapt the template files as to also use (visible) tables for element arrangement. Furthermore, the original system provides a progress bar at the top of the dialog, indicating feedback on the proportion of already processed questions. Such feedback components are currently not yet included in *ProET*—by using JavaScript techniques, and by defining a corresponding component type and respective templates/styles, such or similar interactive feedback elements could be added in the future.

As a second case study, we chose the *Labour Legislation Consultation*, that in contrast to the rheumatic consultation implements a fundamentally different, hierarchy-based interaction style. There, the problem to solve—in this case the question, whether an employment contract was terminated legitimately—is displayed as the top element in the hierarchy (see Figure 4) and its current rating (e.g., established/suggested/excluded) is indicated by its coloring. On the next hierarchical sub-layer, questions are displayed that help to clarify that problem—in the example, the second item from the top "Compliance with form...", and all items on the same hierarchical level. If reasonable, those questions are further subdivided—for example, question "Dismissal was not prohibited..." (third item from the top) is subdivided into eight further questions, that describe more detailed aspects of the parent question. Thus, the user can choose, whether to answer the more abstract questions, or rather the more refined ones. Based on the provided answers, the system calculates ratings regarding the problem statement; those ratings of the sub-questions are accumulated into one rating, that is then presented for the parent question—this propagation proceeds up through the complete hierarchy to the main problem statement/solution.

Figure 4 shows, that this system type is matched well by the tool. Also the necessary interactivity—unfolding sub-hierarchies of questions by mouse-click, coloring the questions depending on the answer, and accumulating/propagating solution states throughout the hierarchy—is supported. One minor difference between the original system and the prototype again concerns the interface design: First, the "+" and "-" signs—indicating whether a question can be further subdivided—are not integrated in *ProET*. Also, the coloring of the questions does not end with the last character of the question as in the original, but spans a defined width.

4.2 ProET: Scope and Limitations

First replication case studies so far revealed the need to extend *ProET* with further components and templates, as to be able to recreate the chosen initial set of knowledge systems completely. Examples are aforementioned feedback components (e.g., progress bars), or further options/templates regarding the general layout (e.g., table-based, or multi-column layouts). Despite such minor shortcomings, *ProET* currently comprises a reasonable set of widgets, as well as exemplary prototype specification files, that enable the near-complete recreation of many knowledge systems, developed by our department in the past. Their replication, as well as adapting the given system specifications and corresponding template/style files for examining alternative designs, was rather unproblematic.

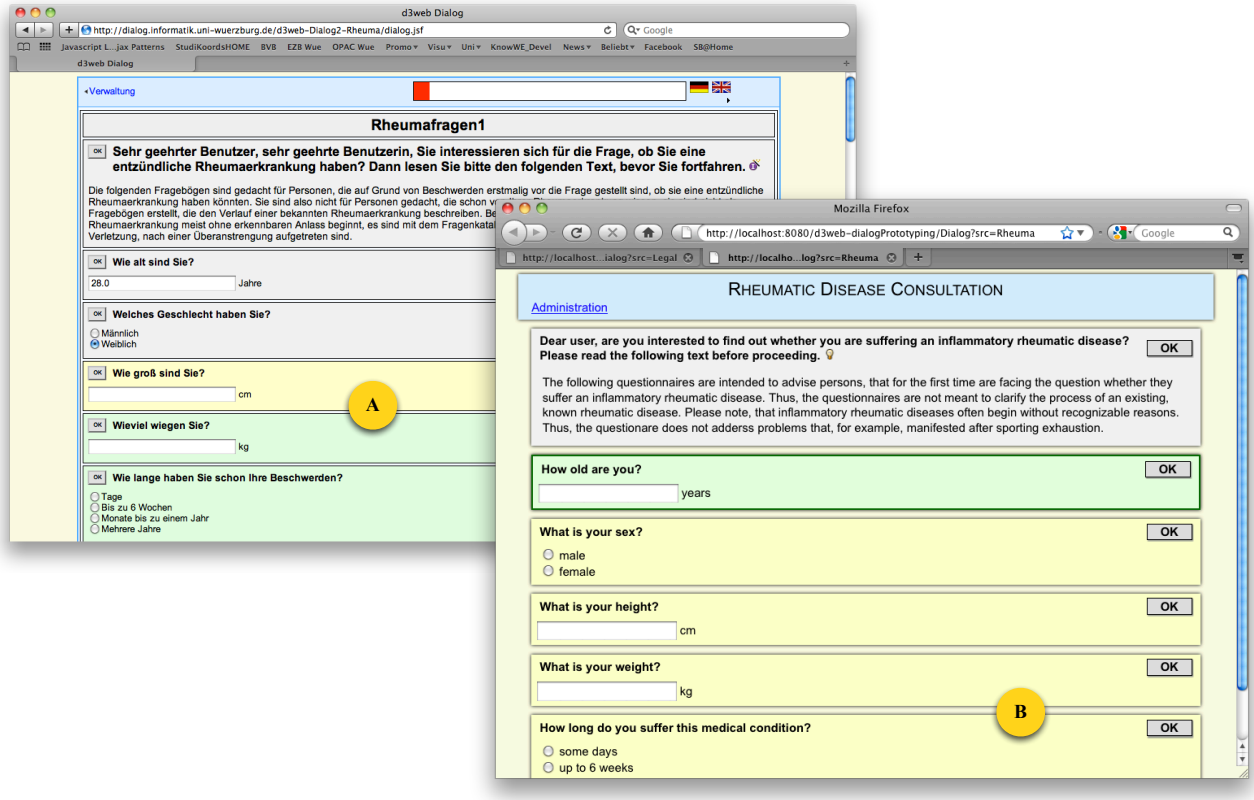

Figure 3: Consultation on rheumatic diseases—an example of a standard questionary-style consultation system. Original system (A, in german) and recreated prototype (B)

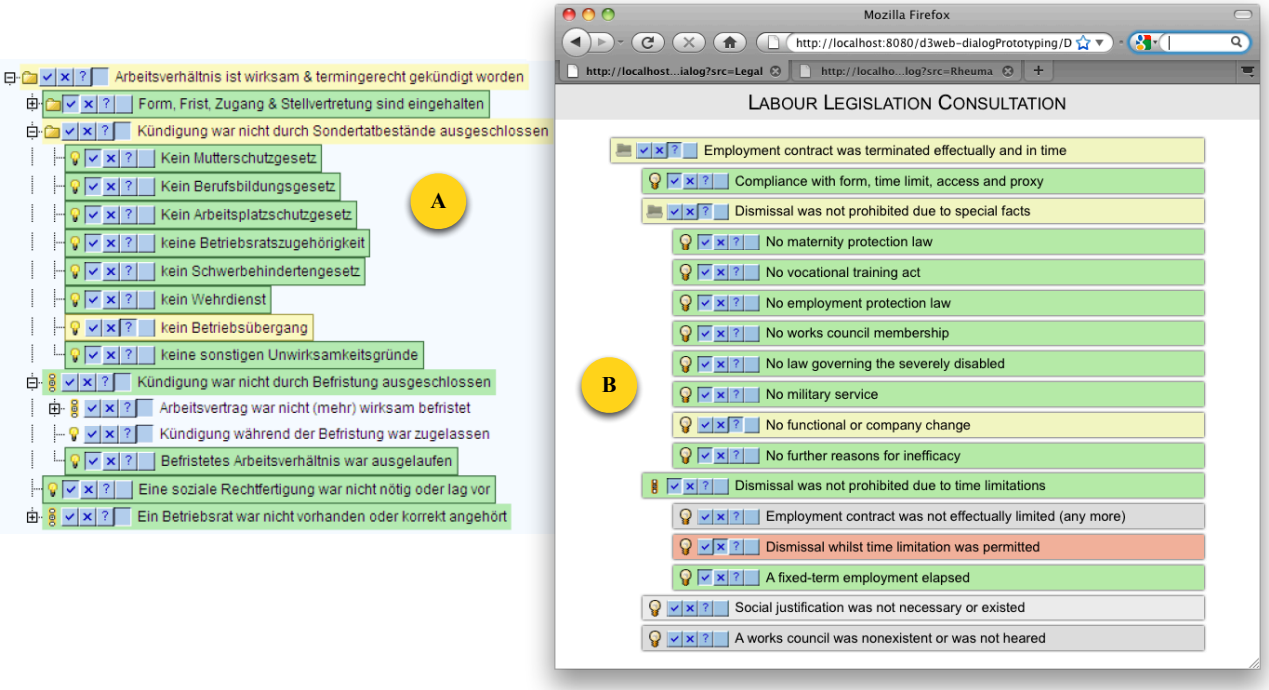

Figure 4: Labour legislation consultation—an example of a hierarchical-style consultation system. Original system (A, in german) and recreated prototype (B).

Yet, the different KS types and by default available widgets that can be prototyped without (major) tool extensions is limited. This is due to the fact that we explicitly chose web-based consultation and documentation systems for defining the initial set of features supported by the tool. Apart from the static widgets, so far also only selected interaction forms are supported. This mainly enables the creation of two basic system styles at the moment: Questionary-based (Figure 3) and navigable hierarchy-based (Figure 4). Those elementary styles can be adapted with regards to various aspects, such as grouping questions into questionnaires, optionally showing side panels that for a direct navigation of the pages, the presentation of solutions and their derivation states, or different forms of designing header and footer elements.

We are aware, that there surely exist other equally relevant knowledge system types and respective designs developed outside our department; yet, a more comprehensive investigation of such external systems, the identification of additional, fundamental KS components, and the appropriate extension of *ProET* is subject of further research.

5 Conclusion

In this paper, we introduced the tool *ProET* for developing knowledge system prototypes. We introduced a tailored process model for pattern-based, prototyping- and usability-integrating KS Engineering, and we discussed potential benefits as well as how the approach can be applied for pragmatically promoting and conducting respective projects.

First experiences with *ProET* revealed the need of its further extension. Apart from systems created by our department, also externally developed knowledge systems will be examined as to identify further relevant components and interactions; a more extensive classification of fundamental elements—in terms of a widget "language"/library—will be defined, leading to an extension of *ProET* as to match that "language". Also, prototyping with ProET is currently purely text-based. Extending the tool to allow also for a more visual form of prototyping—e.g., assembling prototype elements via drag & drop—is another interesting research issue, as this provides the chance to render the prototyping process per se much more intuitive.

Another open question is, whether usability evaluation components can and should be directly integrated into *ProET* (e.g., integrating some tailored logging mechanism to track the "usage" of the prototype). A further idea is incorporating tailored usability guidelines/heuristics into the tool in the form of interactive questionnaires, that can be optionally rendered integrated with the prototype, enabling its rather straightforward evaluation.

Also, the direct linking of the *d3web* toolkit[2] to *ProET* is under way. *d3web* facilitates the development of deployable knowledge bases, thereby supporting various problem-solving methods (e.g., heuristic rules, or set-covering models). This coupling first enables the integration of deployable knowledge bases with *ProET*, permitting an easy investigation, which KS type and corresponding (interaction) design is suitable for a given knowledge base—e.g., developed in the course of actual projects—or also more generally, for a specific knowledge representation. The long-term objective is to extend UI prototypes into productive knowledge systems with no or minimum additional effort.

[2] http://d3web.sourceforge.net/

References

[1] J. Angele, D. Fensel, D. Landes, R. Studer, Developing Knowledge-Based Systems with MIKE, Automated Software Engg. 5 (4) (1998) 389–418.

[2] J. Baumeister, D. Seipel, F. Puppe, Agile development of rule systems, in: Giurca, Gasevic, Taveter (eds.), Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Publishing, 2009.

[3] Bäumer, Dirk and Bischofberger, Walter R. and Lichter, Horst and Züllighoven, Heinz, User Interface Prototyping—Concepts, Tools, and Experience, in: ICSE '96: Proceedings of the 18th international conference on Software engineering, 1996, pp. 532–541.

[4] M. Beaudouin-Lafon, W. Mackay, Prototyping tools and techniques, in: The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003, pp. 1006–1031.

[5] C. Floyd, A Systematic Look at Prototyping, in: Approaches to Prototyping, Springer-Verlag New York, Inc., 1984.

[6] M. Freiberg, J. Baumeister, A survey on usability evaluation techniques and an analysis of their actual application, Tech. Rep. 450, Institute of Computer Science, University of Würzburg, Germany (2008).

[7] M. Freiberg, J. Baumeister, F. Puppe, Interaction pattern categories—pragmatic engineering of knowledge-based systems, in: Proceedings of the 6th Workshop on Knowledge Engineering and Software Engineering (KESE-2010) at the 33rd German Conference on Artificial Intelligence, 2010.

[8] H. Lichter, M. Schneider-Hufschmidt, H. Züllighoven, Prototyping in industrial software projects—bridging the gap between theory and practice, in: ICSE '93: Proceedings of the 15th international conference on Software Engineering, 1993, pp. 221–229.

[9] Y.-K. Lim, E. Stolterman, J. Tenenberg, The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas, ACM Trans. Comput.-Hum. Interact. 15 (2) (2008) 1–27.

[10] J. Lin, J. A. Landay, Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces, in: CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, 2008, pp. 1313–1322.

[11] M. McCurdy, C. Connors, G. Pyrzak, B. Kanefsky, A. Vera, Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success, in: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, 2006, pp. 1233–1242.

[12] M. R. McGee-Lennon, A. Ramsay, D. McGookin, P. Gray, User evaluation of OIDE: a rapid prototyping platform for multimodal interaction, in: EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems, ACM, New York, NY, USA, 2009, pp. 237–242.

[13] T. Memmel, H. Reiterer, A. Holzinger, Agile methods and visual specification in software development: a chance to ensure universal access, in: UAHCI'07: Proceedings of the 4th international conference on Universal access in human computer interaction, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 453–462.

[14] J. Nielsen, Iterative User Interface Design, IEEE Computer 26 (11) (1993) 32–41.