A Knowledge-Based Framework for the Rapid Development of Conversational Recommenders

Dietmar Jannach and Gerold Kreutler

Institute for Business Informatics and Application Systems
University Klagenfurt
Universitätsstraße 65
9020 Klagenfurt, Austria
{dietmar.jannach, gerold.kreutler}@uni-klu.ac.at

Abstract. Web-based sales assistance systems are a valuable means to guide online customers in the decision-making and product selection process. Conversational recommenders simulate the behavior of an experienced sales expert, which is a knowledge-intensive task and requires personalized user interaction according to the customers' needs and skills. In this paper, we present the ADVISOR SUITE framework for rapid development of conversational recommenders for arbitrary domains. In the system, both the recommendation logic and the knowledge required for constructing the personalized dialog and adaptive web pages is contained in a declarative knowledge-base. The advisory application can be completely modeled using graphical tools based on a conceptual model of online sales dialogs. A template mechanism supports the automatic construction of maintainable dynamic web pages. At run-time, a controller component generically steers the interaction flow. Practical experiences from several commercial installations of the system show that development times and costs for online sales advisory systems can be significantly reduced when following the described knowledge-based approach.

Introduction

Customers are increasingly overwhelmed by the variety of comparable products or services available on the market. Unlike in real buying environments where sales experts with adequate experience and knowledge support customers in selecting the optimal product, in the online channel customers are generally not provided with good sales assistance. Thus, online customer assistance is not only a means to reach a competitive advantage by offering an additional value to the customers, in e-commerce it is even a necessity. Web-based sales assistance and recommendation systems are used to guide online customers in their product selection process. In order to be of real value, such systems have to simulate the behavior of a human sales assistant, i.e., elicit the customer's needs and preferences in a personal dialog ([1], [2]) and come up with one or more suitable proposals. In addition, adequate explanations for the recommendation are required in order to increase the customer's confidence in his buy-

ing decision. Such dialogs have to be carried out in a *conversational* style ([3], [4]), i.e. with a mixed-initiative interaction.

In a real world conversation, there are two types of knowledge that an experienced sales assistant exploits. First, there is the basic recommendation knowledge to determine the suitable product according to the customer's needs and preferences. Second, the sales assistant needs to know *how* to acquire the real customer's needs. Intuitively, he will therefore adapt his conversation style to the customer's skills and then ask different questions or offer adequate explanations, depending on the current situation.

The goal of the ADVISOR SUITE project was to develop a framework and corresponding tools for rapidly building intelligent, maintainable conversational recommenders for arbitrary domains. Using an *expert system* is the natural choice for such a problem, as the expert's knowledge is made explicit and stored in a declarative knowledge base that is strictly separated from domain-independent recommendation algorithms. In order to fully exploit the benefits of a knowledge-based system, the expert knowledge about the different conversation styles also has to be formalized. This, however, leads to the problem that the user interface of such an application has to be extremely flexible and dynamic; during an interactive advisory session, the customer's characteristics have to be continuously evaluated based on the answers to different questions, and the dialog flow as well as the presentation style has to be adapted dynamically.

There are some important requirements to be respected in the design of such a recommendation system. First, the dialog pages must be robust against changes in the knowledge base, i.e., the questions to be displayed to the customer, the possible answers, as well as other personalized text fragments have to be dynamically retrieved from the repository. Second, these dynamic pages have to be simple, comprehensible, and maintainable by a web developer who has to adapt the layout using standard HTML and style sheets, such that it is aligned e.g., with the company's corporate design. Finally, there have to be adequate tools and intuitive conceptual models, such that the typically costly process of making the expert's knowledge explicit is simplified.

The software framework presented in this paper illustrates a practical application where several best-practices from the field of Web Engineering (e.g., [5], [6], [7], [8]) are implemented for a specific kind of web applications, i.e., conversational recommenders. In the following sections, the main architecture, implementation details, as well as practical experiences from commercial applications of the system are described and a comparison with other approaches in the area is drawn.

Overall Architecture

Figure 1 sketches the overall architecture of the system. All of the required expert knowledge is stored in a common repository built on top of a relational database system ([9], [10]). The ADVISOR SUITE system comprises a set of graphical knowledge acquisition tools that are used by the knowledge engineer and the domain expert for modeling the recommendation logic as well as the interaction and personalization knowledge. The "user characteristics", i.e., his/her preferences, needs, and wishes are the main pieces of knowledge which are elicited in an advisory dialog either by direct

questioning the online customer or by indirect reasoning. These characteristics determine the products to be proposed as well as the interaction and presentation style.

Based on the modeled information about the dialog, e.g., the definition or the dialog screens, questions, questions' styles or dialog sequences, the *GUI Generation Module* is used to automate the web page development process. Therefore, this module automatically constructs the source code for the corresponding dynamic web pages by using a *template* mechanism which is described in a later section.

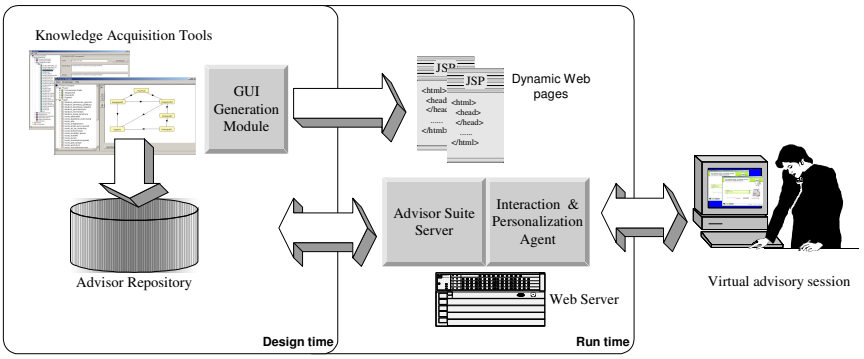

Fig. 1. Architecture of the framework

At run-time, the sales advisory application runs on a standard web server. For each customer session, an *Interaction and Personalization Agent* manages the interaction with its client and forwards the user to the correct dialog screens. The agent also communicates with a *Server Module* that implements the core reasoning logic, e.g., product recommendation based on the customer's characteristics. The Server module itself loads the knowledge from the repository on start-up and also stores information to the repository, like all interaction data, customer properties, session statistics, and other information collected by the interaction agent.

Modeling the sales assistance system

When developing an expert system, elicitation and formalization of the expert knowledge are the critical tasks. As such, it is important to have an adequate and intuitive form of knowledge representation that can be easily understood by domain experts. The whole advisory process in the ADVISOR SUITE system is driven by the customer properties which are modeled as variables that can take one or more predefined values from a given domain. One simple example is the *user expertise*, i.e., the customer could be asked for a self-assessment of his knowledge level at the beginning of the dialog. Corresponding to the variable's domain, possible values are for instance "*beginner", "advanced",* or "*expert"*. The customer's answers (i.e., his profile) consequently influence both the set of products to be recommended as well as the subsequent dialog steps, as different questions for experts and beginners could be defined. Note that in some domains the real user characteristics cannot be acquired based on

self-assessment by direct questioning alone, e.g., when determining the risk class of a customer in investment advisory. For such cases, ADVISOR SUITE also supports indirect reasoning on customer properties based on expert knowledge.

In this paper, we do not go in detail of the constraint-based recommendation algorithms. Basically, the domain expert defines *filtering rules* that relate customer characteristics to product properties like "*If the customer is a beginner and has no adequate financial background, do not recommend investment products with a high risk*", an example taken from the investment advisory domain. For modeling purposes, a classical "if-then" style notation and a high-level constraint language is used. It can also be understood by non-programmers and entered via a graphical context-aware editor (Figure 2). The designed language comprises expressions over customer- and product properties containing arithmetic, relational, logical and set operators.

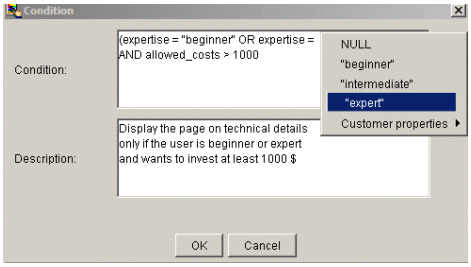

Fig. 2. Context-aware condition editor

After the application of all filter constraints, possibly no suitable products could remain. In this case, we use an algorithm based on Hierarchical Constraint Satisfaction [11] for priority-based filter relaxation. Furthermore, in order to increase the customer's confidence, the sets of applied and relaxed filters are used to construct an explanation of the recommendation. Finally, for computing a personalized ranking of the remaining products that takes the user's preferences and interests into account, a standard multi-attribute utility technique (MAUT, [12]) is applied.

The acquisition of the customer characteristics by direct questioning and indirect reasoning is done in an interactive advisory session. Therefore, these characteristics are also the starting point for designing the personalized dialog flow. In order to simplify this modeling step, we base it on a generic, conceptual model of a sales advisory application and a corresponding graphical notation. There were two driving factors in the design of that conceptual model. First, we have to use a terminology and a notation that is understandable by domain experts, but still has a defined semantics for automating the application construction process in later steps. Intentionally, we did not rely on standard methods for modeling application dynamics like UML State Diagrams [13] or Petri Nets for representation purposes, because from our experiences domain experts are not acquainted with defined technical terms like "state" or "transition". Second, the gap between the model of the application and the resulting web pages and other components has to be kept small in order to be able to directly relate changes in the model to changes in the application.

Figure 3 sketches conceptual model of a sales assistance application in a UML class diagram. The advisory application consists of a set of dialog pages, whereby on

each page a set of questions and their possible answers can be displayed in a given style, e.g., as radio buttons. Note that these questions and answers can correspond both to customer characteristics and product properties, i.e., the customer can also state his requirements on the basis of technical product features. Further, for questions and answers explanatory texts in multi-lingual versions can be defined. In order to model the dialog flow, each page has a set of possible successor pages, whereby for each of them a transition condition over the customer properties can be defined. These conditions are again modelled using the system's constraint language with a context aware editor (see Figure 2) and are evaluated at run-time by the *Interaction and Personalization Agent* that determines the next dialog page. Furthermore, the whole dialog can be organized in phases, which is typically used to provide the customer feedback about the dialog's progress or additional navigation possibilities.

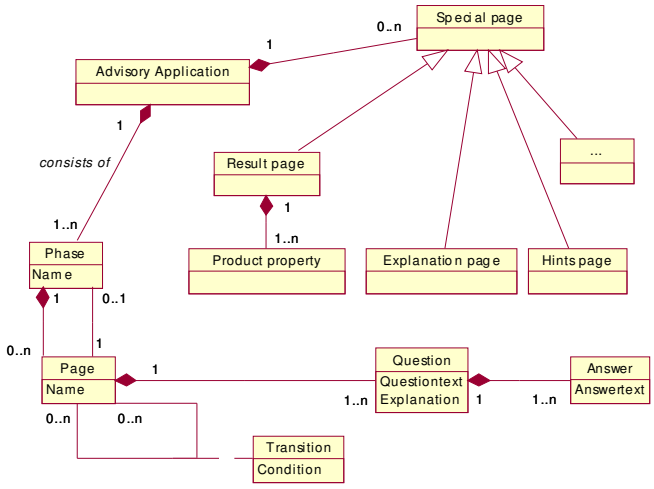

Fig. 3. UML model of an advisory application

In addition to application-specific pages, there are also domain-independent *special pages*, like a page where the results are displayed or where the explanations for the proposal are presented. In most cases, these pages are quite similar for different domains and only differ in the set of product properties to be displayed or in the layout and positioning of the elements. For instance, one special page is the one where *hints* are displayed during the dialog. We allow domain experts to model conditions when the standard dialog flow should be actively interrupted by the system. Typically, this is done when the customer's answers are conflicting, or in situations when an experienced sales assistant would provide additional explanations or hints. This feature is also commonly used for cross-selling and up-selling purposes. The interruption of the dialog depends on the current customer's characteristics and the corresponding conditions that are again evaluated by the Interaction Agent at run-time.

In order to support the domain experts in modeling the interaction flow, graphical modeling tools are integrated in the ADVISOR SUITE system. Figure 4 shows a screenshot of this tool with a simple page flow from the domain of digital cameras.

User interface generation

One of the major goals of the ADVISOR SUITE project was to automate the development process for the web application as far as possible. This feature should allow us to reduce the overall development and maintenance times, guarantee consistent high quality of the web application, and provide rapid prototyping facilities without the need for highly skilled web developers.

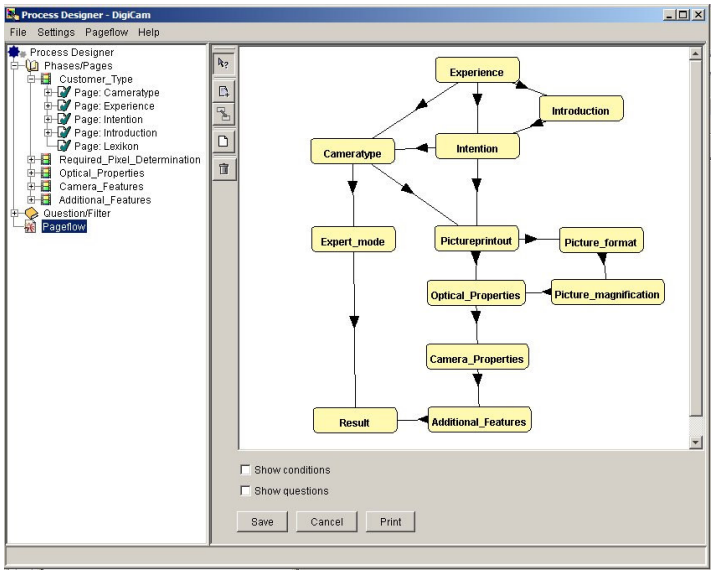

Fig. 4. Dialog modeling tool

The approach taken in the ADVISOR SUITE system relies on two basic techniques. First, we make extensive use of *Custom Tags* [14] in the generated dynamic *Java Server Pages (JSP)*[1]. *Custom Tags* are syntactically similar to standard HTML tags and can be included in the application's HTML code. Internally, however, they implement arbitrary application-specific functionality and behavior. This technique strictly separates static HTML code from Java code or other scripting code that determines the dynamics of the page. Therefore, one consistent "programming" model is used.

The listing fragment in Figure 5 shows how custom tags are included in standard HTML code. In this example, predefined tags are used to display a question corre-

[1] see http://java.sun.com

sponding to a customer characteristic together with the possible answers in an HTML table element. Internally, the custom tag evaluates the current customer's characteristics and retrieves the personalized question text and the possible answers that are defined in the knowledge base and displays the corresponding information, e.g. in the language of the current customer. The tags also transparently manage the communication with the Interaction Agent, e.g. for steering the dialog flow. The implementation details of the tags, however, are completely hidden for the web developer. The ADVISOR SUITE framework provides such tags for most of the functionality typically required in sales advisory applications, e.g., displaying the result products, explanations, or dialog hints. Note that using this technique, no changes in the HTML code are needed, if the contents in the knowledge base change.

The second mechanism exploited for automated application development is based on page assembly and parameterization using adaptable *templates*. This assembly task is performed by the *GUI Generation Module* after maintenance activities on the knowledge base, e.g., when a new dialog page is inserted into the page flow. In our framework, each dialog page consists of a set of predefined areas like headers, navigation area, or an area for displaying a question (see Figure 6). For these areas, small HTML templates with no more than thirty lines of HTML code are provided, that can be adapted by the web-developer, in case that the style or positioning has to be changed or additional HTML content should be included in the page.

```
<advise:question name="$QUESTION_NAME$">
<table class=QUESTIONDISPLAY>
   <tr><td>
      <advise:questiontext/>
   </td></tr>
   <advise:answers>
      <tr><td>
         <advise:radio/> <advise:optiondisplay/>
      </td></tr>
   </advise:answers>
</table>
</advise:question>
```

Fig. 5. HTML fragment with custom tags

In fact, the listing in Figure 5 is such a template for displaying a question with radio buttons for the answers, whereby the radio buttons are arranged vertically in table rows. A typical layout change in the template, for instance, would be a horizontal arrangement of the radio buttons, which only causes a small change in the template. Nevertheless, the placement of questions on pages as well as the display style like using radio buttons is defined in the process modeling tool without changes in the templates. The *GUI Generation Module* then generates one Java Server Page for each page defined in the conceptual model and replaces the placeholders (e.g., $QUESTION_NAME$ in the example) with correct values.

The described template mechanism is robust against repeated generation of pages after maintenance activities, as the changes are done in the templates. In cases when very specific functionality that cannot be generalized to all pages has to be implemented, the generated but still readable pages can be edited. On repeated generation of the application, the changed page can be prevented from being overwritten by the

Generation Module. Note that we also make extensive use of predefined Cascading Style Sheets (CSS) within the standard templates, which allows us adapt the presentation style like layout of the elements as well as positioning independently from the knowledge base content.

Implementation and practical experiences

The whole ADVISOR SUITE framework including the knowledge-based recommendation and personalization engine was built using Java and HTML technology and runs on standard web servers and relational database management systems. The integration of external data sources like pre-existing electronic product fact sheets can be done on the basis of a Java database connectivity (JDBC) interface or an XML-based data exchange format.

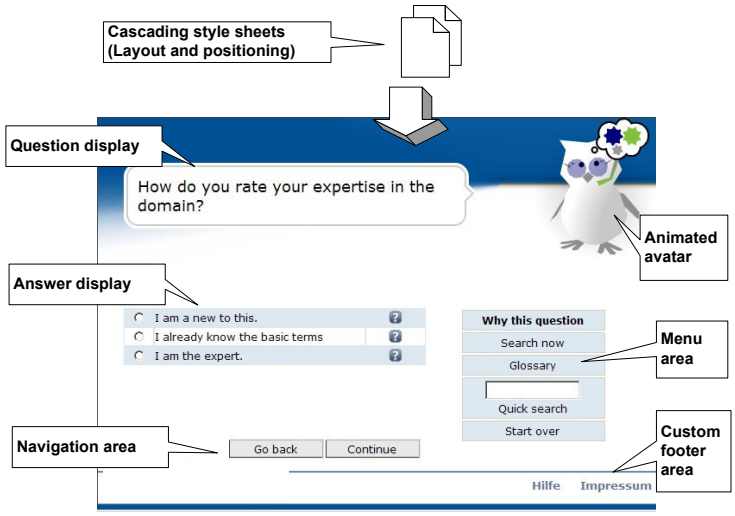

Fig. 6. Structure of a generated page

From the point of view of development, we experienced that the consistent use of technologies significantly increases the quality of the software and the interoperability between the individual components. This holds both for the internal quality of the framework as well as for the generated sales advisory web applications. In order to avoid maintenance problems in the highly interactive web application, we also minimized the usage of client-side scripting technology like *JavaScript*. Furthermore, typical performance issues that potentially arise in Java-based applications are addressed by extensive data caching. In addition, a pre-compilation mechanism for fast processing of filtering rules and personalization conditions increases performance. In a practical setting with an installation on one of Austria's largest e-commerce web

sites[2], around twenty thousand full advisory sessions in the first few days did not cause any performance problems, whereby only standard hard- and software for the Web server was used.

Besides increased customer satisfaction by the value adding service, companies offering online sales assistance systems also profit from the new information provided by the system. As mentioned before, the ADVISOR SUITE framework collects information gathered during the customer dialog, e.g., interaction data, session data, or customer characteristics. Thus, companies can learn about their customers and their preferences and desires, which is necessary for a successful Customer Relationship Management (CRM, [15]). As an example, the information about the customers' self-assessment of the expertise can be useful in order to tailor the online service to this target group. On the other hand, the evaluation of click-behavior of the online users can be helpful to improve the advisory application itself, e.g., it is possible to determine if the dialogue is terminated by the users prematurely. Such an analysis can be supported by advanced Web Mining techniques [16], which will be part of our future work.

Up to now, several commercial sales assistance systems were built with the ADVISOR SUITE framework, whereby the application areas range from technical products like digital cameras or video projectors, to complex domains like investment advisory in private banking, as well as to "quality and taste" domains like fine cigars or vacation planning. Our experiences show that the overall development times (and the corresponding costs) for the sales assistance application are extremely short. For most domains, the initial setup of the knowledge base including the basic design of the dialog can be done in very few workshop days with the help of a knowledge engineer. Although the provided development tools were typically not directly used by the domain expert for the initial setup and prototyping phase, they served as a good basis for communication in these phases. Later maintenance activities, however, required the help of the knowledge engineer only in few cases.

The development times for the final web user interface varied with the specific requirements on the layout that was mostly determined by the existing web site. In some cases, only minor changes in the generated application's layout were required, in other cases more efforts had to be spent for the implementation of domain-dependent, specific application-behaviour or for a smooth integration into an existing web site. Our experiences show that the realization of the advisory service in a "wizard-style" separate application window was well appreciated by the users mostly because the recommendation process can be experienced as a easy-to-follow and focused task. From a technical point of view, easy integration of our system into existing web sites is ensured by the consistent use of standard technologies.

In all cases, however, we found that the maintenance or extension of the generated pages was no problematic issue and could be done by a web developer without advanced programming skills. In fact, by using the described Custom Tags and standard HTML, the generated pages remain small and readable and the developers soon understand how the pages are assembled. Note that specific extensions for a customized behaviour of the application can be incorporated into the application by adding arbitrary *Java Server Pages* code to the generated pages or the templates, whereby also an

[2] with respect to unique daily visits, http://www.geizhals.at

Application Programming Interface (API) for the communication with the advisory server is provided.

As a result, the quality of the web pages of the application remained at a high level, as many tasks like maintenance of multi-lingual version, management and automatic logging of user inputs in the HTTP session, or communication with the web server are already automated. Moreover, the given structure of the generated web application guarantees that the clear separation of the underlying model, the page flow control, as well as the presentation style with style sheets in the sense of the Model-View-Controller concept [17] is consequently obeyed.

Related work

Applications that are built with ADVISOR SUITE belong to the class of conversational recommender systems (see, e.g., [3] and [4]) which have the advantage that the user can be guided in the decision process in a personalized dialogue. When compared to the work of [4], differences can be found for instance with respect to the possible complexity of dialogues, which is somewhat more restricted in our work. Bridge [4], for instance, focuses on a formal "dialog grammar", drawing ideas from Conversational Analysis for defining conversational dialogs. Although more complex dialogs can be modeled, we argue that this formal, domain-independent approach is problematic with respect to knowledge acquisition, because it is harder to understand for domain experts. In addition, when the dialogs become more "natural", some systems suffer from the problem that end users attribute to the system more intelligence than is warranted. Overall, in this paper we have mainly focused on the engineering and maintenance aspects of conversational recommender systems than on the personalization or recommendation functionality itself. Some functional features of our system, however, like the generation of adequate explanations which are made possible by the consistent knowledge-based approach, cannot be found in other conversational recommender systems.

In the field of *Web Engineering*, several approaches have been presented over the last years that aim at applying state-of-the-art Software Engineering practices to the development of such web applications, or extend such common methods to fit the specific requirements of these applications. The main goal is to support a better design, structuring, and understanding of data-intensive web applications by the use of abstractions, corresponding conceptual models, and graphical notations.

The Web Modeling Language (WebML) ([5],[18][3]), for instance, defines a *data model* that extends Entity/Relationship diagrams to capture relevant entities and relationships in the domain; a *hypertext model*, that specifies the content units that make up the page; and *links* that express navigation possibilities. Another approach, HDM/OOHDM (Object Oriented Hypermedia Design, [6]) uses an extended UML [13] notation to model the data view and introduce *navigational classes* (nodes, links and access structures) and *navigational contexts* for the critical part of navigation modeling in web applications in order to separate it from data and interface design. In

[3] see also http://www.webml.org

recent work (e.g., [7]), an even more general architecture for building different families of web applications is introduced. Furthermore, some existing modeling methods are extended in order to support the conceptual design of web applications, for instance UML using stereotypes [19] or the OO-Method [8]. The main focus of these approaches is a smooth integration of web application specific concepts into a general software production process, which enables automatic generation of parts of the code on the basis of the conceptual models.

Our work differs from the above-mentioned approaches insofar, as we do not aim at providing a comprehensive methodology for developing web applications in general, but rather limit ourselves to conceptual modeling and development support for a specific family of applications in the sense of [7], i.e., web-based sales assistance systems. Nonetheless, several concepts of such general *Web Engineering* approaches are instantiated in a particular form in the ADVISOR SUITE system. First, there is a clear separation of the different aspects of the application, whereby the application is not data-intensive, but knowledge-driven. Therefore, the underlying *model* is not a data model in terms of a class diagram describing data structures, but rather a knowledge-base consisting of expert rules over problem variables. The model of the application's dynamics – the dialog flow – is quite similar to the above-mentioned approaches and is in our case based on a model of the application's web pages and navigational links between them. In contrast to general approaches, however, we did not chose a technically oriented notation like *state diagrams*, but we utilize a problem-oriented, proprietary notation for the target users, i.e. domain experts, who usually have not sufficient expertise in standard conceptual modeling techniques.

The separation of the page content and its presentation in our framework is very rigid. In the modeling phase only the phases, pages, and the questions to be displayed as well as the basic presentation template is chosen. The actual layout and presentation style is determined by the style sheets and is not influenced by changes in the dialog flow. Nevertheless, the limitations of the Model-View-Controller concept in the context of interactive hypermedia techniques [7], where the navigational logic is in many cases mixed with interface layout, are addressed by the use of a knowledge-based, generic controller component that dynamically computes navigational links and page successor relations in the context of the current user's characteristics. The usage of one generic dialog model instead of several contextual models also allows us also to keep it at a manageable size.

Because of their targeted level of generality, the automated page and application generation mechanism of ADVISOR SUITE can not be found in the general approaches. However, compared to the WebML approach, the basic structure of the dialog pages can be seen as part of an instance of the *hypertext model* that pre-defines the individual units of which a page consists. The UML model describing the general structure of the generated application (Figure 3) can be regarded as an *instance* of a conceptual model or a typical pattern of a sales assistance application, and could also be expressed in the notation of [18] or [19].

In addition to the work of existing approaches from the Web Engineering field, we also see extended tool support during the whole development cycle as an important issue. The modeling tools provided in the ADVISOR SUITE framework enables rapid prototyping cycles during the analysis (knowledge acquisition) phase, which has shown to be extremely helpful. Moreover, the quality of the resulting applications can

be increased by the usage of properly engineered HTML templates and custom tags. Finally, the presented framework also provides basic tool support for HTML-editing and resource management consistently into the web development process, as it is explicitly mentioned as a focus of the work in [8].

Some correspondence can also be found to the field of *Domain-Specific Modeling* – DSM ([20], [21], [22]). In contrast to general modeling languages like UML, domain-specific modeling languages are based on concepts and the specific semantics of the particular application domain and serve as starting point for automated code generation. Beside that increased automation of the software development process, these approaches have also the advantage that the domain-specific notation and terminology can be more easily understood by the domain experts. The domain-oriented modeling techniques and the code generation capabilities of ADVISOR SUITE can be seen as such an DSM approach that enables domain experts to express their knowledge on a high abstraction level that can be exploited to generate a executable web-based application.

Conclusions

In this paper we have described a framework for rapid development of maintainable conversational recommenders. We follow a consistent knowledge-based approach both for the core recommendation task as well as for the design of a web interface that has to support complex and personalized user interaction. The usage of graphical knowledge acquisition tools, a conceptual model of the application, and automated web page generation based on templates allowed us to significantly reduce development efforts, whereby also the internal quality of the generated web application can be kept at a high level.

Compared with popular approaches from the area of Web Engineering, the work presented in this paper in many facets can be regarded as an implementation of best-practices from different methods with a difference of a knowledge-driven underlying problem and an end-user oriented notation and terminology.

Future work will include data mining techniques to be able to better exploit the information collected during the customer interactions, both for supporting CRM-related analysis tasks as well as to increase the quality of the recommendation process by analyzing user behavior. Further research will also be done towards the generalization of the presented techniques for dialog and navigation modeling for other application domains that require personalized user interactions.

REFERENCES

[1] Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schäfer, R., and Zanker, M.: A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems, *AI Magazine*, 24 (3), Fall 2003, 93-110.

[2] Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schäfer, R., and Zanker, M.: Personalizing on-line configuration of products and services, *Proceedings 15th European Conference on Artificial Intelligence*, Lyon, France, IOS Press, 2000.

[3] Thompson, C.A., Göker, M.H., Langley, P.: A Personalized Systems for Conversational Recommendations, *Journal of Artificial Intelligence Research*, 21, 2004, pp. 393-428.

[4] Bridge, D.: Towards Conversational Recommender Systems: A Dialogue Grammar Approach, *Procs. of the Workshop in Mixed-Initiative Case-Based Reasoning*, Workshop Programme at the Sixth European Conference in Case-Based Reasoning, pp. 9-22, 2002.

[5] Ceri, S., Fraternali, P., and Matera, M.: Conceptual Modeling of Data-Intensive Web Applications. *IEEE Internet Computing*, 6 (4), pp. 20-30.

[6] Rossi, G., Schwabe, D., Esmeraldo, L., Lyardet, F.: Engineering Web Applications for Reuse, *IEEE Multimedia*, 8 (1), 2001, pp. 20-31.

[7] Jacyntho, M.D., Schwabe, D., Rossi, G.: A Software Architecture for Structuring complex Web Applications, *Journal of Web Engineering*, 1 (1), October 2002, pp. 37-60.

[8] Gomez, J., Cachero, C., Pastor, O.: Extending a Conceptual Modelling Approach to Web Application Design, *Proc. of the 1st International Workshop on Web-Oriented Software Technology*, Valencia, Spain, June, 2001.

[9] Jannach, D.: Advisor Suite – A knowledge-based sales advisory system, *Proc. of the 16th European Conference on Artificial Intelligence – 3rd Prestigious Applications Intelligent Systems Conference*, Valencia, Spain, 2004, pp. 720-724.

[10] Jannach, D., Kreutler G.: Building on-line sales assistance systems with ADVISOR SUITE, *Proc. of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE 04)*, Banff, Canada 2004, pp. 110-116.

[11] Schiex, T., Fargier, H., Verfaille, G.: Valued Constraint Satisfaction Problems: Hard and Easy Problems, Proc. of *International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montreal, Canada, 1995, pp. 631-639.

[12] von Winterfeldt, D., Edwards, W.: Decision Analysis and Behavioral Research, Cambridge University Press, Cambridge, UK, 1986.

[13] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, Addison-Wesley, 1998.

[14] Goodwill, J.: Mastering JSP Custom Tags and Tag Libraries, Wiley Publishers, 2002.

[15] Berson, A., Smith, S., Thearling, K.: Building data mining applications for CRM. McGraw-Hill, New York, 2000.

[16] Kosala, R., Blockeel, H.: Web mining research: A survey. *SIGKDD Explorations* 2 (1), 2000, pp. 1-15.

[17] Krasner G.E., Pope S.T.: A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. ParcPlace Systems Inc., Mountain View, 1988.

[18] Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a Modeling Language for Designing Web Sites, *Computer Networks*, 33, 2000, Elsevier, pp. 137-157.

[19] Conallen, J.: Building Web Applications with UML, Addison Wesley, Reading, 2000.

[20] Gray, J., Rossi, M., Tolvanen, J.-P. (Eds.): Domain-Specific Modeling with Visual Languages, *Journal of Visual Languages & Computing* 15 (3-4), June-August 2004, pp. 207-330.

[21] Tolvanen, J.-P., Kelly, S.: Domänenspezifische Modellierung, *ObjektSpektrum*, 4/2004, pp. 30-35.

[22] MetaCase Corp.: Domain-specific modeling: 10 times faster than UML. White Paper, available online at http://www.metacase.com/, August 2004.