Modeling Submit/Response Style Systems with Form Charts and Dialogue Constraints

Dirk Draheim and Gerald Weber

Institute of Computer Science, Freie Universtität Berlin
draheim@inf.fu-berlin.de

Abstract. Form-Oriented Analysis is an approach tailored to the modeling of systems with form-based, submit/response style interfaces, a distinct and ubiquitous class of software systems. Form-Oriented Analysis models the system interface with a bipartite finite state machine and relates it to a layered data model. The paper explains the main visual artifact of our technique, the form chart, and introduces dialogue constraint writing. Model decomposition is explained. The analysis technique is firmly based on existing well-understood analysis notions and techniques, and consequently extends these methods.

1 Introduction

In this paper we present Form-Oriented Analysis, a new analysis technique for a distinct and ubiquitous class of interactive software systems. This class covers well-known form-based applications ranging from typical Internet shops through supply chain management to flight reservation systems. We give a precise definition of the considered class of software systems and have coined the term *submit/response* style applications for this system class.

Submit/response style applications are such applications that present to the user a page that offers information as well as a number of interaction options, typically forms. If the user has filled out a form and *submits* the form the system processes the data and generates a *response* page. This response page again offers different interaction options to the user. We model such a submit/response style application in a way that will turn out to be well suited for such systems, namely as a bipartite state machine, which alternates between presenting a page to the user and processing the data submitted by the user. This bipartite state machine is depicted in the key artifact of Form-Oriented Analysis, the form chart. Form-Oriented Analysis describes then, how to annotate this bipartite state machine with constraints, which specify the behavior of the system. The definition submit/response style is not intended to cover all kinds of software systems, but to single out a well-defined and important class of systems. There are of course other interactive software systems that do not follow this metaphor. In many software systems such as text editors or drawing tools the interaction with the system does not proceed by submission of forms that lead to a new page. Instead, the current screen is constantly updated as the interaction proceeds. However,

R. Meersman and Z. Tari (Eds.): OTM Workshops 2003, LNCS 2889, pp. 267–278, 2003.

submit/response style applications form a ubiquitous class of important systems, which justifies the development of an analysis method specifically designed for this type of systems.

Technically, submit/response style applications can appear as modern web applications or as client/server applications or of course as classic mainframe applications. However, we deal in this paper with the *analysis* of such systems, and the particular technical representation shall be transparent for the functional specification as a result of the analysis phase, hence we want to achieve a *specification* independent from *implementation*. It is therefore the key goal of this paper to establish a high level view on this type of systems, in which we abstract from the underlying technology and focus on the inherent properties of submit/response style systems. Not every problem is amenable to a solution by a form-based system. But if a system is intuitively thought of as being accessible by a submit/response style interface this gives an important starting point for the problem analysis. In the analysis technique proposed here, called Form-Oriented Analysis, we will give a powerful approach to system modeling by understanding the system along its usage through a submit/response style interface. This interface model in Form-Oriented Analysis is an abstract interface; it is a conceptual tool for the understanding of the system. But it can be thought of as a working prototype of the actual application interface. Hence Form-Oriented Analysis is a technique for modeling a system along a prototypical submit/response style interface.

The restriction of Form-Oriented Analysis to submit/response style applications allows us to employ the clear semantics of submit/response style interfaces within the analysis phase. Hence a model obtained in form-based analysis benefits in its formal strictness and semantic clarity from the restriction to this interaction style. Form-Oriented Analysis covers the area of analysis which is typically called the functional specification.

Form-Oriented Analysis uses mainly visual artifacts for modeling. But in contrast to other visual modeling techniques we understand these artifacts mainly as a visualization of information, which also could be given in a textual representation. This flavor of visualization is important for Form-Oriented Analysis since it is a technique designed for tight integration into a suite of code based tools.

This paper explains the form chart, which is the main contributed artifact of Form-Oriented Analysis. The other form-oriented diagram types, page diagram and form storyboard [9], are important informal predecessors of the form chart, which highlight specific aspects. Page diagrams offer a natural conceptual basis for modeling of submit/response-style software system. Form storyboards are designed with respect to informal communication between domain experts and system analysts. The special way that signatures of server actions are visualized as forms make form storyboards able to serve as high-level prototypes. Form storyboards can be transformed into form charts without structural friction. Form charts are used for rigorous software system specification. Our method

offers a simple yet powerful composition mechanism for artifacts, called feature composition.

Form-Oriented Analysis does not prescribe any process model. Of course, the different degree of formality of the different artifacts as well as the feature composition mechanism hints at a process like intuitive succession of diagrams from informal to formal, but it is important to realize that this is by no means necessary. Since the semantics of the diagrams is completely independent from any process definition, the diagram is basically neutral with respect to its use in a process of whatever kind. However, our method can be used easily with agile methodologies due to similarity between form charts and the actual code architecture. With the feature composition paradigm, form charts can easily cope with changing requirements.

In section 2 we present the central artifacts of Form-Oriented Analysis. Dialogue constraints are discussed in section 3. Composition of artifacts is addressed in section 4. Tool-support for Form-Oriented Analysis is described in section 5. We give a dedicated discussion on selected related work in section 6.

2 Form Charts and Model

Form charts introduce into the system model one of the major conceptual cornerstones of Form-Oriented Analysis: the system is seen as a bipartite state transition diagram. The bipartite state machine is the basic structure of form charts. In this view the system is alternating between two kinds of states. The first kind of states corresponds to the pages of the system. These states are called client pages. The system remains in such a client page state until the user triggers a page change. In that moment the record with her previous input is sent to the system. The second kind of states represent the system actions in response to page changes and are therefore called server actions. These states are left automatically by the system and lead to a new client page.

We demonstrate Form-Oriented Analysis for an example seminar online registration system. Our example system is a form-based seminar registration system as it is suited for a single course. The screen shots of the system are shown in Figure 1. The front page shows the students registered so far and contains links to the different interaction options. New students can register themselves. From the homepage, students already registered can change or delete their registration. Each link leads to a new page specific for the chosen option.

The form chart has the task to make the analysis model amenable to formal constraint writing and coupling to the semantic data model, and it is therefore accompanied by two other diagrams, first the semantic data model and second the data dictionary mediating between both. Furthermore a textual document containing formal constraints has to be seen as attachment to the form chart. The document bundle consisting of form chart with attached constraints, data dictionary and semantic data model comprise the *form-oriented specification* of the system. A complete specification of a system is often a valuable goal, but in many cases it may not be practically achievable. Our method allows the

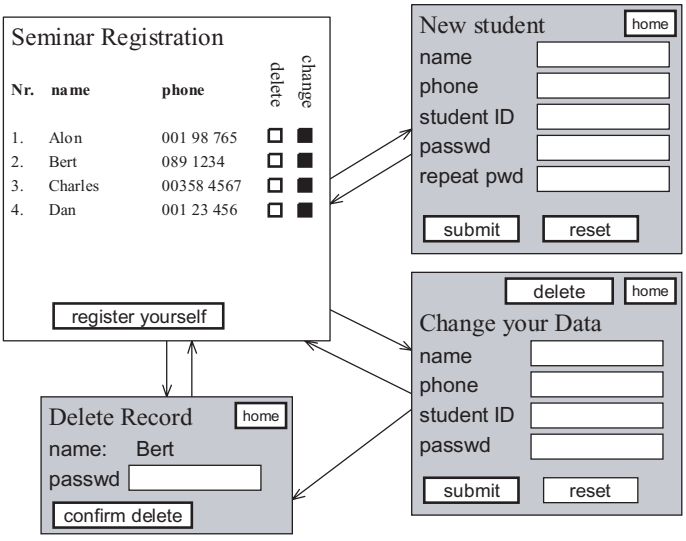

Fig. 1. Overview of the screens of an online seminar registration system.

modeler to create a complete specification, but of course it is usable for partial specification as well and therefore gives the modeler the freedom to choose the degree of precision which seems appropriate for the project.

The form chart as shown in Figure 2 contains the bipartite state machine. Server actions are depicted as rectangles and client pages are depicted as bubbles. In the form chart only the names of the states and transitions appear. The form chart also contains the start marker. The second new artifact type, the data dictionary, is shown in Figure 3. The data dictionary contains types and is therefore a class diagram in the terms of modern modeling languages like the UML. However, the data dictionary types are of a special kind of data types, namely algebraic data types. Instances of these types are immutable values. The types can have structure, but only a hierarchical structure, namely composition. They represent sent messages, comparable to written and sent documents. Remaining in that metaphor, once you have sent a letter the content is unchangeable. In the data dictionary there must be a message type for each form chart state, and it must have the same name, except that the initial letter is lower case in the form chart, but upper case in the data dictionary.

The last diagram in the bundle that forms the specification is the semantic data model. This diagram is the conceptual data model that specifies the system state. Pure data record specifications which are needed by both, the semantic data model as well as the data dictionary, are put in a separate part of the data dictionary, the business signature repository. In our example the semantic data model is rather simple and consists mainly of the class holding the student information. The semantic data model is connected with the data dictionary again through the dialogue constraints, but also through so called opaque references.

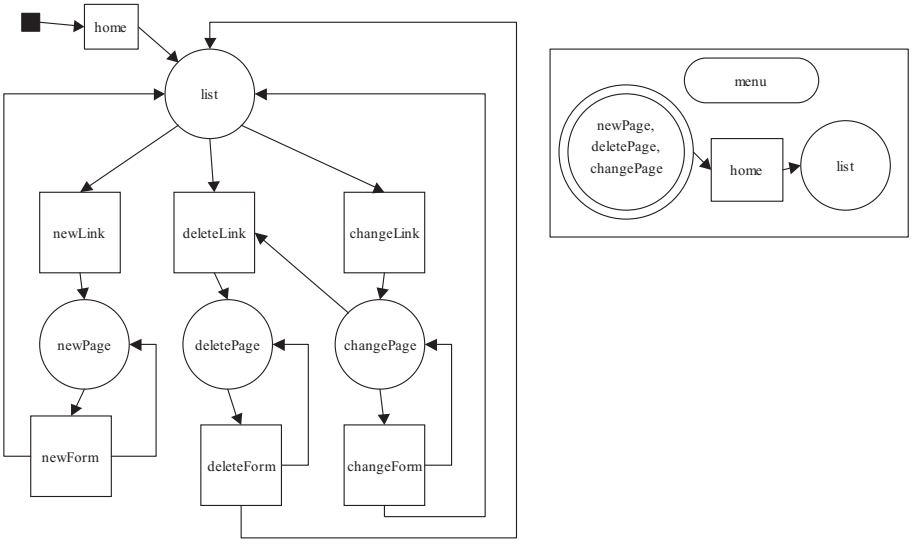

Fig. 2. Form chart of the seminar registration system.

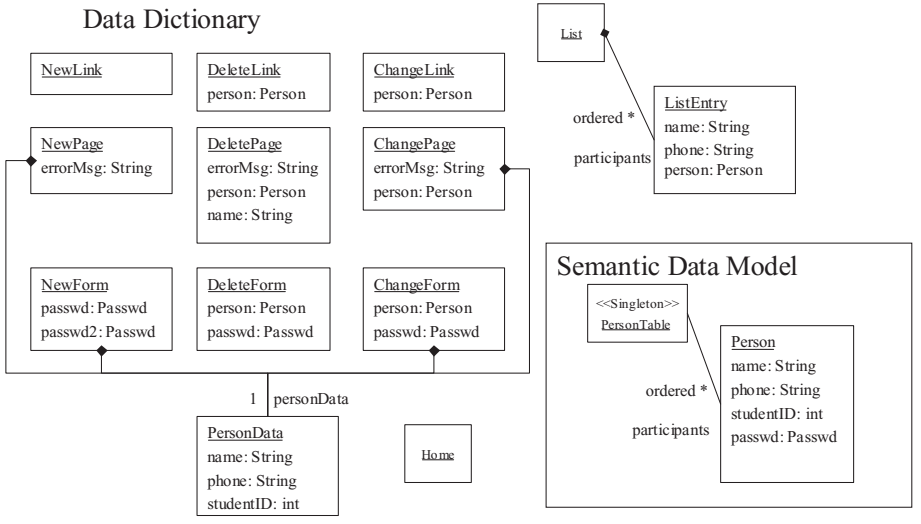

Fig. 3. Semantic data model and data dictionary

Take the client page `list` as an example. The message type `List` contains a list of different `ListEntry` objects that contain only the information presented to the user. This `ListEntry` submessage contains an attribute of type `Person`, the class from the semantic data model. Such a reference from the data dictionary to the semantic data model is called opaque reference. As one can see, if one follows the message types associated e.g. with the delete subdialogue, this reference is

passed along the dialogue and hence specifies, which person object is subject to the deletion process. The reference is passed along the form chart, yet the reference is opaque in the sense that only through certain operations that again access the semantic data model the content of the person object can be accessed.

The whole semantic data model forms a single data abstraction module with possibly as many opaque reference types as it contains classes. The opaque references are therefore the border of the data dictionary. The reference itself is part of the message, but not the referenced object. Therefore the object can change without violating our demand, that messages are unchangeable.

3 Dialogue Constraints

The message represents the signature of the state of same name. Each time this state is entered, a new message of this type has to be provided. We also specify signatures for the client pages. These client page signatures represent the information shown on the page. The page content is immutable. A page shows the same content to the user until she triggers a page change and therefore gets a new page, although possibly from the same type. Page interaction, i.e. user input in forms is not considered a change of the page content, but preparation of a new message. The fact that now the data dictionary contains the information shown on pages as well as the information sent back to the system as part of a page change is important with respect to the specification of so called dialogue constraints. Indeed one of the main advantages of form charts is that it allows elaborate constraint writing. We want to be able to express e.g. that the data record the user chooses for deletion must have been presented on the page. Such a constraint is called client output constraint. It is written in the following style.

```
list to deleteLink {
clientOutput:
source.participants.person->
  includes(target.person)
}
```

As we see in this example, we need the signature of the client page as well as the signature of the server action, called source and target, in order to set both in relation to each other. There are a number of different types of dialogue constraints, and they form together the dialogue constraint language, DCL for short. The DCL constraints are typically written in an attachment of the form chart, although in principle they can be written into the form chart diagram itself.

The Dialogue Constraint Language DCL introduces special purpose constraint types, which are shown in Figure 4. Transitions from client pages to server actions, page/server transitions for short, host two kinds of constraints, namely enabling conditions and client output constraints. An enabling condition

specifies under which circumstances this transition is enabled, based on the state during the last server action. The enabling condition may depend on the current dialogue history. The data submitted from a client page is constrained by the client output constraint. Server actions host server input constraints. They are server action preconditions in an incompletely specified system, they must be transformed to other conditions. Transitions from server actions to client pages, called server/page transitions for short, host flow conditions and server output constraints. The flow conditions specify for each outgoing transition, under which condition it is actually chosen. The server output constraint determines which information is presented on the client page that follows in the sequel. The client input constraint is a constraint on the information on the client page, which is independent from the server page.

The constraints in the form chart are written in a variant of OCL [18]. For this purpose OCL is enriched by new contexts and key labels with appropriate semantics due to the needs of dialogue constraint writing. Consequently data modeling is done with the pure data kernel of UML, whereby we distinguish message types in the so-called data dictionary from persistent data within the semantic data model. Persistent data can be accompanied by ephemeral session related data. The system functionality is seen as side effects of server actions. It may be specified in the context of the server action, but it typically will be structured by functional decomposition.

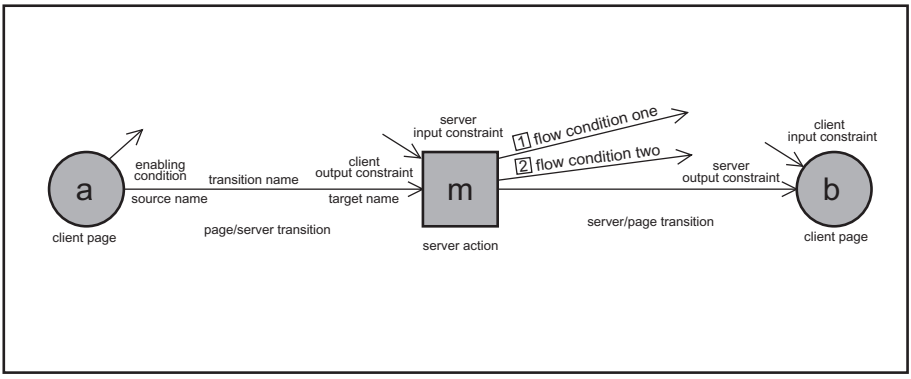

Fig. 4. Form chart notational elements

4 Feature Composition

Feature composition is introduced as the composition mechanism for form charts. The graph structure of a form chart has been specified as being a bipartite directed labeled multigraph. Every sub graph of the form chart is called a feature

chart. Two feature charts are combined by graph union. A form chart decomposition is a collection of feature charts in such a way that the combination of the feature charts yields the complete form chart.

The perhaps most intuitive explanation, why feature composition is possible and meaningful in Form-Oriented Analysis is the inverse operation, feature decomposition. A complete form chart has a uniquely stable semantics: If page/server edges, i.e. interaction options are removed, the data integrity is not endangered. Certain usages of the system may of course become impossible, if one removes key interaction options for the system. But the semantic data model is not corrupted by such operations: the system remains stable, if it was stable before. As a consequence the form chart covers system behavior that is inherently stable against runtime customizations.

The composition of the analysis model is of course especially important with respect to the task of expressing preferences and priorities in the system specification, as well as to enable the discussion of alternatives and trade-offs between them.

4.1 Compatibility Issues

There are some rules for the composition of two features. The rules follow from the fact that the features to merge must be subgraphs of one single form chart. First no node is at the same time client page in one graph and server action in the other. Nodes of the same name must have the same data dictionary type, because different features are different form charts over the same data dictionary and model.

If two features are combined, the constraints have to be compatible. If in a feature composition step a server action receives server/page transitions from different features, the flow condition numbers in both features must be different in order to be merged into a single order unless they are mutually exclusive. The server/page transition without flow condition has to be the same in both features, or one of the features should have no server/page transition without flow condition.

4.2 Hierarchical Feature Decomposition

A form chart can be decomposed in a hierarchical manner. The result is a tree of chart decompositions. Decomposition makes the form chart manageable. It is a tool for organizing the form chart artifact during the analysis phase. The feature hierarchy as such is not semantically relevant for the specification. Every combination of feature charts, even from different levels of the tree, yields a correct sub graph of the form chart.

4.3 Menu-Like User Interface Parts

An important special case of feature composition is the modeling of menu-like options, i.e. interaction options, which are offered on many, perhaps even all

pages. A new notation element for this purpose is the state set, that is depicted by a double lined state icon. It is annotated by a list of state names and serves as shorthand notation for these states. The example in Figure 5 shows page sets. An edge between two state sets of say m client pages and n server actions represents the complete bipartite graph $K_{m,n}$ between the elements of the state sets. A feature chart may be annotated as menu. Then the page/server transitions contained in this feature must not be contained in the main form chart or its decompositions. Affected states may reference the respective menu feature chart by an explicitly given name. Figure 5 shows how the described mechanism fosters readability of system interfaces with menu-like user interface parts. Another notation flavor is to give the state set a single name, and to reference the page set in its member states. The menu construct is used in the form chart of the seminar registration system in order to model the home button.

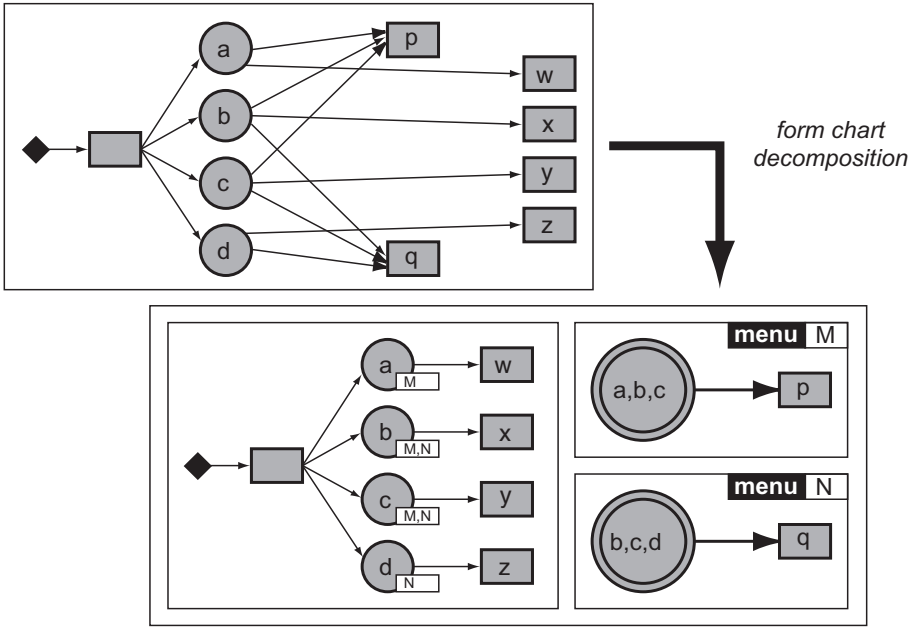

Fig. 5. Modeling menu-like user interface parts

5 Tool-Support for Form-Oriented Analysis

The model obtained in Form-Oriented Analysis can be transformed without impedance mismatch into an interface design and an implementation based on well-established technologies for web interfaces like server pages [6]. GENTLY is a proposed specification language for web-based presentation layers that provides

a textual format of form charts. The forward engineering tool GENTLY [8] and the design recovery tool JSPick [7] both exploit the specification language GENTLY. The GENTLY tool generates a complete prototypical dialogue based on Java Server Pages from a high-level system description in GENTLY. The JSPick tool generates high-level system descriptions for Java Server Pages based web presentation layers in a GENTLY dialect.

6 Related Work

Structured Analysis [15] is a very successful approach to both business modeling and system modeling that is still used in practice. It combines hierarchical data flow diagrams, sum-of-product data specification, local functionality specification and later [21] entity-relationship diagrams. The method is deliberately ambiguous with respect to the semantics of the several notational elements of the data flow diagrams and therefore heavily relies on the intuition of the modeler. Structured Analysis does not at all take into account driving forces of the solution domain.

The use-case driven approach to object oriented software engineering had deep impact. From the beginning [13] to state-of-the-art versions [14] of this approach the recommended human computer interface specification techniques exclusively target the modeling of GUI's. Furthermore the approach still lacks real world examples clarifying the meaning of use case specifications and how these can be exploited during system design and implementation.

State diagrams has been used for a long time in user interface specification [19], [10], [12], [20]), partly with the objective of user interface generation [4]. All of these approaches target user interface specification only at a fine-grained level, in our terminology concerning page interaction. Another early approach [11] targeted the modeling of push-based, form-based systems like the already disussed single-user desktop databases.

Within the UML Community the Discussion about dealing with the user interface is still underway [3]. In [5] a visual language for presenting user interfaces is proposed. The new artifacts are basically visualizations of page components. The method is tightly coupled with the use case driven approach. In our view, the diagrams do not reach the intuitive clarity of our proposed artifacts. A stereotype framework specifically for web applications is presented in [2]. This approach allows to model the design level concepts appearing during web site development with a typical web application framework. For this purpose the Conallen approach uses a set of stereotypes. The approach targets rather design than analysis.

Schwabe et al. presented a diagrammatic tool for representing web interaction [16], [17]. The diagrams are called user interaction diagrams (UID). They resemble page transition diagrams without server actions. Very restricted and very specific annotations are placed on the transitions concerning required selections by the user.

The aim to reduce the necessary navigation primitives is adressed in WebML [1], a visual language for conceptual modeling of complex web sites [1], in which all concepts are defined visually as well as in XML. WebML offers icons for page elements for composing web sites, e.g. catalogue pages and single item views. The WebML approach can be seen as an advanced and customizable successor of model driven interface generators.

7 Conclusion

Form-Oriented Analysis is an analysis technique for submit/response style applications. This class can be seen as a characterization of typical enterprise applications, including e.g. web applications. We model a submit/response style application with bipartite finite state machines, layered data models and dialogue constraints. Form charts are given by rigorous semantics and rules of usage. Our analysis technique is firmly based on existing well understood analysis notions and modeling techniques and consequently extends the state of the art in an important application domain: our analysis method is tailored to the class of submit/response style applications, but not designed as an analysis technique for all kinds of software systems. This strategic decision allows Form-Oriented Analysis to fit optimally to submit/response style applications and to provide added value for the analysis of such systems.

References

1. S. Ceri, P. Fraternali, and S. Paraboschi. Web Modeling Language(WebML): a modeling language for designing web sites. In *Proceedings of the 9 th. International World Wide Web Conference*, pages 137–157. Elsevier, 2000.
2. J. Conallen. Modeling Web Application Architectures with UML. *Communications of the ACM*, 42(10):63–70, 1999.
3. J. F. E. Cunha and N. J. Nunes. Towards a UML Profile for Interaction Design: The Wisdom Approach. In *Proc. UML'2000*, LNCS 1939. Springer, 2000.
4. P. P. da Silva. User Interface Declarative Models and Development Environments: A Survey. In *Proceedings of 7th International Workshop on Design, Specification and Verification of Interactive Systems*, LNCS 1946, pages 207–226. Springer, June 2000. Limerick, Ireland.
5. P. P. da Silva and N. W. Paton. UMLi: The Unified Modeling Language for Interactive Applications. In *Proc. UML'2000*, LNCS 1939, 2000.
6. D. Draheim, E. Fehr, and G. Weber. Improving the Web Presentation Layer Architecture. In *In Proceedings of APWeb 2003 - The 5th Asia Pacific Web Conference*, LNCS. Springer, 2003. to appear.
7. D. Draheim, E. Fehr, and G. Weber. JSPick - A Server Pages Design Recovery Tool. In *CSMR 2003 - 7th European Conference on Software Maintenance and Reengineering*. IEEE Press, 2003. to appear.
8. D. Draheim and G. Weber. Specification and Generation of JSP Dialogues with Gently. In *Proceedings of NetObjectDays 2001.* tranSIT, September 2001. ISBN 3-00-008419-.

9. D. Draheim and G. Weber. Storyboarding Form-Based Interfaces. In *INTERACT 2003 - Ninth IFIP TC13 International Conference on Human-Computer Interaction*. IOS Press, 2003. to appear.
10. M. Green. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5(3):244–275, 1987.
11. P. J. Hayes. Executable Interface Definitions Using Form-Based Interface Abstractions. *Advances in Human-Computer Interaction*, 1:161–189, 1985.
12. R. J. K. Jacob. Using Formal Specifications in the Design of a Human-Computer Interface. *Communications of the ACM*, 26(4):259–264, 1983.
13. I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.
14. I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
15. D. Ross. Structured Analysis: A language for communicating ideas. *IEEE Transactions on Software Engineering*, 3(1), 1977.
16. P. Vilain, D. Schwabe, and C. S. de Souza. Modeling Interactions and Navigation in Web Applications. In *Proceedings of 7th International Workshop on Design, Specification and Verification of Interactive Systems*, LNCS 1921, pages 115–127. Springer, October 2000.
17. P. Vilain, D. Schwabe, and C. S. Souza. A Diagrammatic Tool for Representing User Interaction in UML. In *Proc. UML'2000*, LNCS 1939. Springer, 2000.
18. J. Warmer and A. G. Kleppe. *The Object Constraint Language*. Addison-Wesley, 1999.
19. A. I. Wasserman. A Specification Method for Interactive Information Systems. In *Proceedings SRS - Specification of Reliable Software*, IEEE Catalog No. 79 CHI1401-9C, pages 68–79. IEEE, 1979.
20. A. I. Wasserman. Extending State Transition Diagrams for the Specification of Human-Computer Interaction. *IEEE Transaction on Software Engineering*, SE-11(8):699–713, 1985.
21. E. Yourdon. *Modern Structured Analysis*. Yourdon Press, Prentice-Hall, 1989.