An Introduction to Form Storyboarding

Technical Report B-02-06

Dirk Draheim and Gerald Weber
Institute of Computer Science
Free University Berlin
email: {draheim,weber}@inf.fu-berlin.de

March 2002

Abstract

We introduce form storyboarding, a method for eliciting, specifying and communicating functional requirements for applications with form based interfaces. The method encompasses a visual language for the documents which have to be created, and a set of proposals for activities. The visual language is based on bipartite state transition diagrams.

Contents

1 Introduction

In this paper we introduce form storyboarding, a method that is an amalgamation of requirement elicitation, specification, and high-level UI prototyping. Form storyboarding is applicable to systems with form based interfaces, and addresses requirements on business functionality. We do not consider non-functional requirements. Form storyboarding, the requirements gathering method, is seamlessly integrated with form oriented analysis [2], an analysis method. Form storyboarding yields an artifact, the form storyboard, which is a complete description of the system interface. Form storyboarding encourages the requirement engineer to create the basic structure for the functional requirement specification according to the elements of the form storyboard.

The form based interface is the poor cousin of the human-computer interface community. This did not reduce the practical importance and ubiquity of this kind of interfaces. Time has come to state precisely the abstract concept behind form based interfaces, which has its subtle non-trivialities, and sum up the experience with them. We first give an in depth analysis of the considered class of interfaces and its implications for the requirement specification process. Then we develop the notation of form storyboards and explain the proposed modeling activities.

2 Form Based Interfaces

Many applications like today's enterprise applications use an interface type which can be characterized as form based, submit/response style interfaces.

- form based refers to the fact that these user interfaces collect user input in the style of input forms. Input forms can be understood as a metaphor, namely as a direct translation of paper forms into human-computer interfaces. Input forms are composite structures of input elements.

- submit/response style refers to the screen update policy, namely that all screen updates have to be triggered by a dedicated user action. The user actively submits data or a query, instead of passively watching a constantly updated view.

This type of interface is not tight to any specific technology. On the contrary, the same characteristics can be found in many technologies, e.g. HTML-based clients, main frame terminals, even certain parts of GUI based dialogues: the secondary dialogues in document applications, like the print dialogue, follow this paradigm. An important class of systems of this type are systems with ultra-thin clients. Ultra-thin clients are interface tiers that do not contain business logic themselves. Ultra-thin clients cache the user interaction on one page in the client layer. In this respect they differ from certain terminals, which forward each user interaction and can be considered completely stateless. Systems with ultra-thin clients are typically multi user systems. Ultra-thin clients fit neatly into transactional system architectures. Later we will argue that the caching of data is not solely a solution domain concept, but an interface feature. In the following we will use form based interface as the shorthand notion for the described class of interfaces. Form based interfaces show at each point in time one page, which offers the user a collection of interaction options. Interaction options are input fields of forms and such buttons that trigger a page change. These buttons will be called submit buttons. Many concepts in concrete technologies can be subsumed under these concepts. For example in HTML, links can be seen as submit buttons in textual style. If a page contains a list of submit buttons, then the choice of one of these buttons for submission must be seen as an additional input value: a list of submit buttons is equivalent to one submit button combined with a choice list. In the following we argue that form based interfaces are not only a technical legacy or architectural necessity due to scalability issues, but a valuable general interface metaphor. The concept of form based interfaces is here to stay, since they present the optimal solution in a number of areas, where they are used.

2.1 Two Staged Interaction Paradigm

The difference between input fields and submit buttons lead to a strict two level interaction paradigm. User interaction is separated into interactions on one page and interactions that result in a page change. Interaction on one page, called page interaction, consists of filling out input elements, changing the focus between input elements, resetting the form and other interactions. These user interactions are only temporal and do not change the system state. This allows to abstract from page interactions when modeling the complete interface. Only the other kind of interaction does affect the system, namely page changes. Page changes are either submitting a form or clicking a link, whereby submitting a form is the more general concept: a link can be seen as a form consisting solely in the submit button. If the user triggers a page change, the result of her interactions with the respective form is processed by the system, her interaction with other forms on the page is lost. The recognition of the two staged interaction paradigm leads to the following key insight about requirements specification for form based interfaces: there is no need to specify the fine grained interface behavior on one page. Page interactions are well understood and can be offered by a general browser, which is able to interpret a declarative page description and provide a sufficiently powerful standard page interaction paradigm. This explains the success story of concrete technologies like the HTML form concept together with the web browser as a standard ultra-thin client. Hence specifying form based interfaces is reduced to specifying page changes.

2.2 Cognitive Advantages

Form based interfaces have clear advantages for the self explanatory character of a system. The usage of the system is intuitive, since it is guided by the paper form metaphor. The difference between temporary input and submission, or "sending", is intuitive and fosters the user's understanding of the system. The form based metaphor has a client tier structure in its own, without fixing an implementation. The two classes of interactions structure the work of the user into the work intensive frequent page interactions and the punctual and atomic interactions of the "serious" kind, namely the page changes which also happen to be the conclusion and separation of logically disjoint bunches of work. The submit/response style character brings the user in command of the timing of her system usage. It protects her from irritating disruption of her work by incoming information, as will be explained in section 2.5.

2.3 Interaction in Form Based Interfaces

In form based interfaces the submission of a form is an operation that has exactly the semantics indicated by the metaphor. In computer science terms the submission can be modeled as a method call [1], namely the submission of an actual parameter list with a method name. The actual parameter list is given by the contents of the input elements, the method name is best represented by the form title. The form itself can therefore be seen as an editable method call. As a result of a page change, a new page will be presented to the user. This page contains information and new interaction options for the user, namely new forms.

2.4 Ultra-thin Client Forms vs. Other Forms

Form-like interface layout can be found not only for ultra-thin clients but also in other systems, like desktop databases. However, these are fundamentally different concepts. In desktop databases, a form-like input mask is an editable view of persistent data. Once a field is edited, the corresponding field in the persistent data is changed immediately. In general therefore there is no need for a submit procedure, instead there is a concept of buttons, where a button is a parameter free command. These kind of dialogues perform screen updates in a push style, meaning that the system pushes the new content on the screen continuously. An early presentation of such a paradigm can be found in [5]. In this kind of user interfaces forms could also be called views.

Concerning multi user systems on the other hand, the push style is not applicable to conflict resolution between different users. Updates by other users must not interfere with the current work of a single user.

3 Form Storyboards for Interface Specification

We propose form storyboards, a bipartite extended class of state transition diagrams, tailored for the modeling of form based interfaces. A state transition diagram is a directed labeled graph. The nodes are states of the system, the edges are transitions between the system states. Between two states there can be several edges distinguished by different labels.

3.1 Modeling User Interfaces with State Transition Diagrams

Due to the fact that page interaction is well understood, the main task in modeling the dynamic behavior of a user interface is the modeling of page changes. From this perspective, the user interface remains in the same state as long as the same page is presented to the user. The state changes whenever the user triggers a page change. This full abstraction from page interactions is the key advance of the proposed method beyond other state transition diagram approaches. The system can in principle show an indefinite set of pages. In order to model the interface as a finite state diagram, one has to identify a finite set of page sets, each representing one page. Take for example a catalogue of an online bookstore. The number of categories is likely to change over time. The favorite model would have only one state in the state transition diagram which represents all categories. The information which precise category is shown, is not modeled in the diagram. It may be allowed to mention that the states chosen for the diagram will often match more or less the finite set of different page implementations.

Before we explain form storyboards we introduce a more coarse grained kind of diagram, called page diagrams, which follows the choice of states as explained above. Page diagrams only represent the visible states of the system. They provide a valuable shorthand notation for the corresponding form storyboard.

3.2 Page Diagrams

Page diagrams take a natural finite set of states, as explained above. Naïve page diagrams without precise semantics are often found in practice. The user interface is in one particular state of the page diagram, as long as the user performs page interactions. The edges from this state in the page diagram represent the possible page changes that the user can trigger. Our running example is a seminar registration. On the homepage the current participants are listed. A new participant can register herself, as long as she is not already registered (Fig.1).

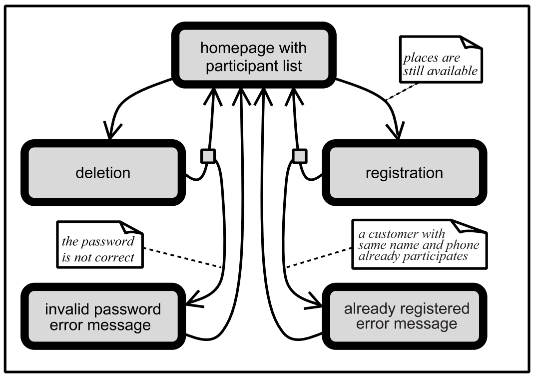

Figure 1: Page diagram

The example page diagram shows one of the important new features of our proposed page diagram formalism: branching edges. They are necessary, because the target page of a page transition may depend on the server action in response to the user input. Error pages are the most ubiquitous example, as is shown in the example: depending on the user input, either the regular subsequent page or an error page is presented, like in the case of an invalid password. In the page diagram this must be indicated as a branching edge, since both pages, error page as well as ordinary response page, are triggered by the same user command. Page diagrams give an intuitive notion, but do not offer a self explaining semantics. A precise semantics for branching edges is gained by form storyboards.

3.3 Form Storyboards

We introduce our new diagram concept called form storyboards (Fig.2). They pay tribute to the intuitive notion that the system usage is an interplay of alternating user action and system action. The states of the diagram are partitioned into page states or user input states and server states. The system state alternates between page states and server states. Hence these diagrams are bipartite graphs. The set of page states is identical to the set of states of the page diagram. The page nodes are denoted as rounded rectangles, the server states are denoted as square rectangles. However, the server states are only temporal: the user action triggers a state transition into a server state, but after processing the system automatically leaves the server state and returns into some page state. In the register action of the running example, the system leads over either to an error state or to the ordinary page state.

The page diagram is the result of collapsing all server nodes to little squares and eliminating those with only one outgoing edge, i.e. uniting them with the unique target page node.

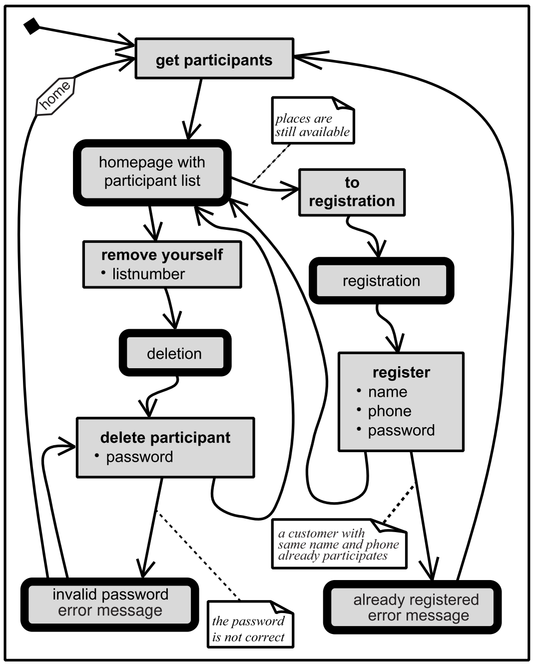

Figure 2: Form storyboard

3.4 Signatures for Server States

Server states are entered by page changes, which we have identified as method calls. We want to include the signature of the method into the diagram. The method signature is written into the square representing the server state. The parameter list is written in the style of a form, so that stakeholders that are not computer scientist have an immediate intuition of the diagram semantics. At the same time the notation fosters the understanding of the developer that a form submission is a method call and that the form is an editable method call. Consider the server state "remove yourself" in the example form storyboard. This server state demonstrates the unification of forms and links. The server state represents a set of links on the page, namely one link for each participant. As explained earlier, this set of links is equivalent to a choice list with all participant numbers and a single submit button. Therefore the server state has one parameter, namely the chosen participant number.

The form storyboard yields a convenient and structured representation of the interaction possibilities on one page. A page state together with its accessible server states represents the interaction options the user can choose. Therefore the evolving form storyboard can be seen as high-level, non-executable prototype, which abstracts from interface layout details, but provides a solid basis for discussion, and form storyboarding turns out to be especially well suited for team work, too. Consider now the form storyboard for the running example, the seminar registration system. Each state can have multiple ingoing edges from different states of the other kind. This means that a server method can be called from different pages, and on the other hand the same page can be produced from different server methods. Furthermore, two states can be connected by more than one edge, if these edges are labeled or annotated differently. Edges from a page node to a server node are called page/server edges, edges from a server node to a page node are called server/page edges. Only page/server edges can be labeled, labels are depicted as hexagonal boxes on the edge. Server/page edges can be annotated with conditions, under which the respective transition will be chosen by the system. These conditions can be annotated in a language of choice. Page/server edges can be annotated with conditions, under which they are enabled, like the link from the homepage to the registration page in the example. In form oriented analysis [2] the informal constraints from the form storyboard can be translated into a formal dialogue constraint language, an extension of OCL.

In form storyboards, exceptions which are caused by exceptional multi-user interference should not be modelled. An example is the case that the user enters the registration procedure at a time where places are still available, but at the time of his submit the last place is occupied. Such a multi-user-exception should be included into the form storyboard only if there is an increased importance to discuss such a case with the domain expert. Multi-user-exceptions have to be dealt with in depth in form oriented analysis.

3.5 Feature Driven Approach

Form storyboarding uses only one kind of formal artifacts. All documents are form storyboards that are understood as subdiagrams of the complete diagram. The completeness of the diagram can be declared locally for each node. Local completeness is given if a diagram shows for one node already exactly those outgoing edges which will be present in the final diagram. Local completeness is depicted by a tickmark in the node. An important artifact is a single feature. A feature contains a small subset of the form storyboard relating to a special aspect of the system. Feature artifacts have a tag containing the annotation "feature".

The domain expert or requirements engineer can denote every subset of the system interface that appears as a single feature in a separate diagram, even not connected parts of the form storyboard can be placed in the same diagram, if they are received as expressing the same feature. An important feature in the running example is the "already registered participant" topic (Fig.3). Form storyboarding is first and foremost a feature driven approach. The whole form storyboard can be gained by collecting single features. Crucial for the simplicity of the form storyboarding method is the fact that diagrams can be combined in an easy operation, by building the union of both diagrams and identifying nodes and edges with the same name.

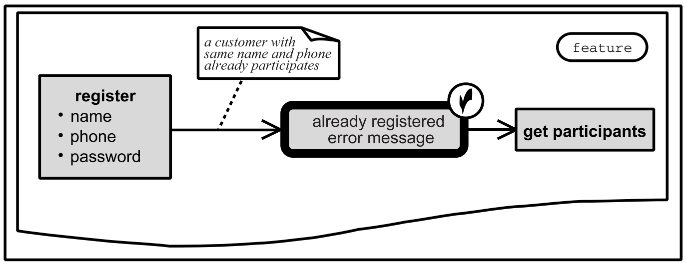

Figure 3: Feature artifact

3.6 Use Cases Reconsidered

Use cases are defined as typical usages of the system. In form storyboards, the basic flow and the alternative flows of a use case correspond to paths in the diagram. The complete diagram can be considered as the compact notation of the set of all use cases. In the form storyboarding approach every use case is a feature.

4 Requirements Elicitation

The ultra-thin client form based metaphor is fundamentally different from other well known UI metaphors, like the desktop metaphor. Namely, the form based metaphor implies a precise semantics of user interaction as outlined before: page interaction is volatile, page change is data transmission and changes the system state. On the other hand, desktop a.k.a. direct manipulation metaphors give only a vague analogy and no precise semantics, e.g.: does drag and drop mean "copy" or "move"? Domain experts understand completely the concept of paper forms. Since paper forms can be translated precisely into system functionality, domain experts can give precise semantics for ultra-thin clients. Therefore form storyboarding encourages that domain experts and requirements engineers discuss system features along the form based metaphor, and the properties are depicted as form storyboard feature artifacts by the requirements engineer. In form storyboarding the requirements engineer does not have the task to translate the semantic concepts of the domain expert into semantics of form storyboarding. Instead, she only has to mediate the notational questions. Therefore form storyboarding has no mismatch between the domain experts view and the specification documents.

4.1 Form Storyboarding Activities

Form storyboarding is not tightly coupled to a specific process for requirement elicitation. Instead form storyboarding offers proposals for activities which in themselves increase the quality of the requirement specification. Form storyboarding is a feature driven approach, not a use case drive approach. The complete form storyboard is gained by collecting all features of the system. Use cases may be used as heuristics for finding features, but are not required. The activities relate to the validation of found features, and to the local systematic search for features. Form storyboarding starts with the development of a number of feature artifacts. The features to be depicted can be gained by various ways, e.g. during brainstorming or by reverse engineering of legacy systems. Each feature can be discussed in its formal notation. If informal search for new features is completed, the systematic search for features is started. The single activity is local search on a single node in the form storyboard. The participants of the requirement process check for a single node, whether the outgoing edges represent exactly the desired interaction options, and check the server response to these options. If the completeness of the description is assured, the node may receive a tickmark. By traversing the form storyboard global completeness is assured.

The form storyboard cannot only be included as a document and diagram into the software requirement specification SRS. From a form storyboard a structure for section 3 of the SRS can be generated. This structure can best be matched with the organizing style "response", as it is given in [6].

5 Related Work

State transition diagrams have early been used in requirements engineering with respect to user interface modeling [3] [7] [11] [12]. An extension of the state transition diagrams are statecharts [4]. Our approach is distinguished from these approaches through:

- The identification of the two-staged interaction paradigm: page interaction and page change.
- The identification of the bipartite structure and the fundamental differences between server states and page states.

The two staged interaction paradigm allows the abstraction from the fine grained page interactions, which are well understood and can be offered by a standard tool, the browser. Hence for form based dialogues, the named other state transition diagram approaches are identified as being too fine grained. Furthermore the system view and dialogue style of these approaches is out of date. User interface modeling is dominated by the discussion of model based user interface development environments (MB-UIDE) [10]. However, again these approaches, when applied to form based dialogues, work on the level of page interactions, which is inadequately fine grained. A current school in requirements engineering is the use case driven approach. HCI specifications as defined in the several versions of this approach [8][9] specifically target the modeling of GUI-based dialogues. The approach does not yield a clear layering of user interfaces into page interactions and page changes.

6 Further Directions

6.1 Textual Format for Form Storyboards

For form storyboards there exists a human readable textual format, called Angie. This language is the completely redesigned successor to the language Gently, which was an early textual specification language for form based interfaces [1]. In Angie the graphical information is incorporated as a special comment. Based on Angie an automatic prototype generator and interpreter is under development. Formal consistency can be defined on Angie and formal semantics can be given to form storyboarding conveniently, even for sophisticated concepts like history or menu techniques found in today's web dialogues.

6.2 Advanced Topics

Functions reserved for certain user classes can be denoted as annotations on page/server edges. The variable actor represents the actor's set of roles, the roles are constants (e.g. "admin actor").

For layout reasons, edges can be split. The stubs end in dotted lines. The starting stub gets an annotation "To: <state>". The terminal stub gets an annotation "From: <state>". This method can be used to decompose complex diagrams.

6.3 Generalization of Form Storyboarding

Form storyboards has been designed to fit the needs of modeling form based interfaces. It then turned out that the form based system metaphor has its own justification. In further work we will discuss to what extent the approach is valuable for modeling other kinds of systems, too. For example, system interfaces may be form based just up to some graphically presented information or up to some system pushed data. It is even possible to generalize the notion of form, guided by the two stage interaction paradigm. Then a form is a bunch of data that is gathered interactively and submitted atomically, but there is no insistence on the resemblance of a form's input capability to a paper form, i.e. it is abstracted from a form's visual presentation. Even in systems which have an overall logic that doesn't adhere to the submit/response paradigm, for example systems that are dominated by an event notification mechanism, insulated use case bundles may be identified which are amenable to be specified by form storyboarding. The models of all these systems will benefit from a separation in form based parts and auxiliary parts.

7 Conclusion

The form based interface paradigm is here to stay. Often ultra-thin clients are the interface of choice. We have introduced form storyboarding, a method for eliciting and specifying requirements on interfaces.

- Form storyboarding offers an adequate abstraction from page interactions.
- The method has a single type of artifacts, namely form storyboards, which are bipartite state transition diagrams.
- Form storyboards can describe features of every desired complexity, they allow simple integration of partial artifacts into the complete form storyboard.
- Form storyboards may serve as a high level, non-executable prototype.

The main contribution of this work is to translate the results about the abstract form based interface into a requirements specification approach.

References

[1] D. Draheim, G. Weber, "Specification and Generation of JSP Dialogues with Gently", Proceedings of NetObjectDays 2001, tranSIT, Ilmenau, September 2001, pp. 3-13.

[2] D. Draheim, G. Weber, "Form Charts and Dialogue Constraints", Technical Report No B 02-08, Intitute of Computer Science, Free University Berlin, March 2002, http://www.inf.fu-berlin.de/inst/pubs/tr-b-02-08.ps.gz

[3] Mark Green, "A Survey of Three Dialogue Models", ACM Transactions on Graphics, vol. 5, no. 3, ACM, 1987, pp. 244-275.

[4] D.Harel, "Statecharts: a Visual Formalism for Complex Systems", Science of Computer Programming, Elsevier Science Publishers B.V., 1987, pp. 231-274.

[5] P.J. Hayes, "Executable Interface Definitions Using Form-Based Interface Abstractions", Advances in Human-Computer Interaction, vol. 1, Ablex Publishing Corporation, New Jersey, 1985, pp.161-189.

[6] IEEE Std 830-1993, Recommended Practice for Software Requirements Specifications, Software Engineering Standards Committee of the IEEE Computer Society, New York, 1993.

[7] R.J.K Jacob, "Using Formal Specifications in the Design of a Human-Computer Interface", Communications of the ACM, vol.26, no. 4, ACM, 1983, pp.259-264.

[8] I. Jacobson, "Object-Oriented Software Engineering: A Use Case Driven Approach", Addison-Wesley, 1992.

[9] I. Jacobson, G. Booch, J. Rumbaugh, "The Unified Software Development Process", Addison-Wesley, 1999.

[10] P. Pinheiro da Silva, "User Interface Declarative Models and Development Environments: A Survey", Proceedings of 7th International Workshop on Design, Specification and Verification of Interactive Systems, LNCS Vol.1946, Springer-Verlag, Limerick, Ireland, June 2000, pp. 207-226.

[11] A.I. Wasserman, "A Specification Method for Interactive Information Systems", Proceedings SRS - Specification of Reliable Software, IEEE Catalog No. 79 CHI1401-9C, IEEE, 1979, pp. 68-79.

[12] A.I. Wasserman, "Extending State Transition Diagrams for the Specification of Human-Computer Interaction", IEEE Transaction on Software Engineering, vol. SE-11, no. 8, IEEE, 1985, pp. 699-713.