Form Charts and Dialogue Constraints

Technical Report B-02-08

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

March 2002

Abstract

Form oriented analysis is an approach tailored to the modeling of systems with submit/response based interfaces. Form oriented analysis models the system interface as bipartite state transition diagram and relates it to a semantic data model. We introduce dialogue constraint writing based on an OCL extension. Model decomposition is explained and different grades of completeness for form oriented analysis models are defined. We outline the integration with form storyboarding, a compatible requirements engineering technique.

Contents

1 Introduction

We introduce form oriented analysis, an approach to the modeling of certain ubiquitous systems with a form based, submit/response interaction paradigm. Form based systems or system parts are frequently found in e-commerce, enterprise applications and other transactional systems. Form based user interfaces are not a legacy technology, but the adequate interface type for the named application domains.

Form based interaction is submit/response type, two staged interaction. The interaction is divided into page interaction, which is temporary and logically local to the client until a submit is performed, and page change, i.e. a submit action. Forms and links can be conceptually unified. Only page change can affect the system data state. Hence the model is two tiered already on the analysis level. The system state in this view does not include the client state, which is the browser's state. The advantage of this software system paradigm is that the client is well understood, independent from the application. Other types of software may also use form like interfaces, but are not submit/response style, e.g. desktop databases and spreadsheet applications found in office suites. They have a single staged interaction paradigm, in which each change is directly a change of the system data state.

Submit/response style user interfaces date back to mainframe applications, but today classic web technology with HTML forms is the most prominent example. Throughout the paper, the approach is exemplified by web interfaces without compromising independence from technology. An HTML form calls a server side script. The called script can be considered a remote method with an input type signature. An HTML form is a user editable method call in general. CGI and its several wrappers like e.g. Java Servlets support transmission of string tuple streams only, hence ignoring the type of the data that is given by the business logic. Form oriented analysis offers strong typing instead. The other submission capability is the HTML link, which can call server side scripts with the same parameter marshalling mechanism as HTML Forms. An HTML link is like a special non-editable HTML form with only hidden parameters.

Form oriented analysis abstracts from page interaction and views a page change always as a method call. In form oriented analysis strong typing is maintained at the system interface.

2 Basic Form Oriented Analysis

In this chapter a succinct description of the main modeling elements of form oriented analysis is given (Fig.1). The chapter presents the basics of form oriented analysis, issues of structuring and refining models are addressed in later sections. In form oriented analysis the system interface is modeled as a bipartite state transition diagram which is annotated with declarative dialogue constraints. These dialogue constraints are written in terms of a data model and additional server side functionality.

2.1 Overview of the Analysis Model

The user interaction with the system, called dialogue in the following, is a sequence of interchanging client states and server states. A client state presents information to the user and offers several capabilities of entering and submitting data. The client state is called client page in the following. By submitting data the dialogue changes into a server state. In the server state submitted data is processed and depending on the current system state the generation of a new client page is triggered, i.e. the server state is left automatically. Submitting data is considered calling a method, the data being an actual parameter. Therefore the server state is called server action in the following. The transition to a client page is again considered a method call, this time executed automatically from the server. Every state has a one-to-one correspondence to a method; in the case of a client state this has to be understood as a purely declarative unit. The method calls are non-returning.

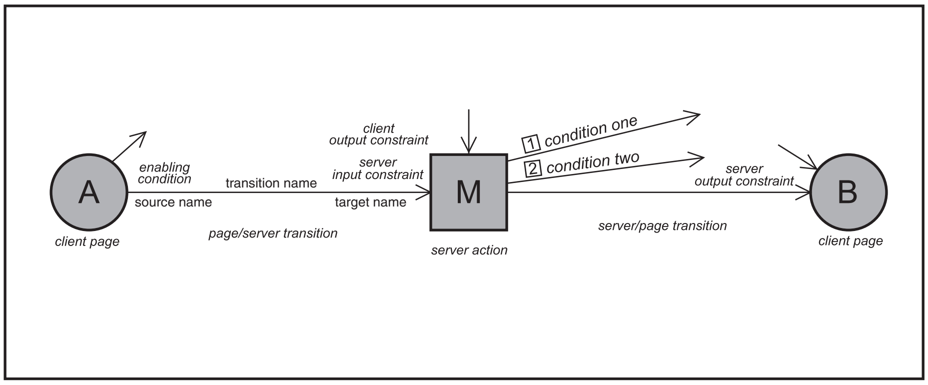

Figure 1: Form chart notational elements

Client states, server states and transitions between them form a bipartite transition diagram. This viewpoint leads to intuitive visual models and furthermore will foster future considerations on advanced topics like transactionality and web based system architecture. The state transition diagrams used in form oriented analysis are called form charts.

Transitions from client pages to server actions, page/server transitions for short, host constraints for expressing enabling conditions, client output constraints and server input constraints. An enabling condition specifies when a transition is enabled, possibly depending on the current use case, more precisely the current dialogue history. Data submitted from a client page is constrained by a client output constraint. Server input constraints are server action preconditions in an incompletely specified system. Transitions from server actions to client pages, server/page transitions for short, host flow conditions and server output specifications. The flow conditions specify which of several outgoing transitions is actually chosen. The output specification determines which information is presented on the client page that follows in the sequel.

The constraints that annotate the state diagram are written in OCL [21]. For this purpose OCL is enriched by new contexts and key labels with appropriate semantics due to the needs of dialogue constraint writing. OCL has been chosen as a basis for the resulting dialogue constraint language despite of its lack of formal semantics [15]. Main arguments are the rich terminology introduced with OCL, its clear informal semantics and most important its usability and expressibility concerning e.g. navigations compared to other alternatives for data type annotation languages [8]. Consequently data modeling is done with the pure data reduct of UML, whereby persistent data is distinguished from ephemeral session related data and web signature type specifications through stereotyping. The system functionality may be defined in terms of additional server-side functionality. In form orientation, this functionality is specified with structure charts with respect to the emerging data model.

2.2 States

In the form chart, client pages are depicted by bubbles, server states are depicted by rectangles. Every state, i.e. every client page and every server action must be given a name. Every state has a signature, which is introduced as an OCL Type. It is defined by a UML class in the data model. This defining class must have the name of the respective state and must be stereotyped as web signature. The web signature is the signature of the state as a method. Every ingoing transition of a state must be a method call with the same signature. Because the signature is combined in one single parameter for every state the term superparameter is introduced. The notion of superparameters eases constraint writing, for example every superparameter has the state name as default name and additionally one or more context-dependent names that are explicitly introduced by the modeler. In accordance with the objective of writing purely declarative dialogue constraints, superparameters must be understood as deep-shallow unchangeable. The superparameter must not be in any way mistaken as parameterized state in the sense of expressing an internal multiplicity of the state.

2.3 Client Pages and Client Output Specification

The signature of a client page serves as abstract description of the information presented to the user. Form oriented analysis does not address layout specification. Beyond the provided information a client page offers one or more data submission capabilities to the user. In HTML/HTTP a group of links may be considered as a single data submission capability under the following conditions: several links that conceptually belong together establish one data submission capability provided that they target the same server side script and send actual parameters for the same set of CGI parameters. Such a link group is like a form with a selection list. Following one of the links is like choosing one list item and submitting the form. Link groups can be found in today's active web sites, for example if the choice between several articles is presented to a customer.

Every page/server transition specifies that the respective client page has a submission capability that calls the respective server action and provides a superparameter. A client page can be read as a collection of forms and links determined by all its targeted server actions. In a form chart a page/server-transition may be context for OCL constraints. These constraints are either output specifications, a server input constraints or enabling conditions, distinguished by an appropriate label. Note that a transition may be labeled with a transition name, source name and target name by the modeler. If not explicitly provided, these names are derived in the obvious way from the names of the involved states.

A transition without client output specification represents a data submission capability that is completely editable by the user. A client output specification is a constraint on the actual parameters that must be ensured by the client page. Actual parameters that are constrained by a client output constraint must either not be editable, but must be correctly provided as hidden parameters or a client-side dynamic watchdog mechanism, be it hand coded or generated [2], must prevent data not fulfilling the constraint from being submitted. An important usage of client output constraints is relating object selection to an information bunch presented to the user, as demonstrated in the following example (Fig.2). Consider a webshop page presenting the user a list of articles by showing a list of links of article names. Article numbers are not shown to the user, but are used as hidden parameters in the links. Clicking a link will trigger the generation of a page that presents to the user detailed information about the selected article identified by the transmitted article number and offers further dialogue options. This is a system that fulfills the specification given by the analysis model comprising Fig.2 and the following constraint:

```
viewArticlesTOviewDetail
output:  viewArticles.presented->exists(number = viewDetail.id)
```

In the above constraint the transition target name `viewDetail` refers to the actual parameter that will be transmitted to the server action. The source name `viewArticles` refers to the actual parameter of the client page. The explanations of server input specifications and enabling conditions are given after explaining server actions.

2.4 Server Action

A server action processes submitted data. Outgoing transitions lead to client pages. The transitions are annotated with flow conditions which are logically mutually exclusive OCL-expressions. For one of the transitions the flow condition may be omitted, having the semantics of an "else" clause. As an alternative to ensuring logical exclusiveness the modeler may number the outgoing transitions to enforce an evaluation order. Based on the flow conditions exactly one of the outgoing transitions is determined after server action processing. The client page which is targeted by this transition is now generated. Thereby the server action provides an actual parameter which

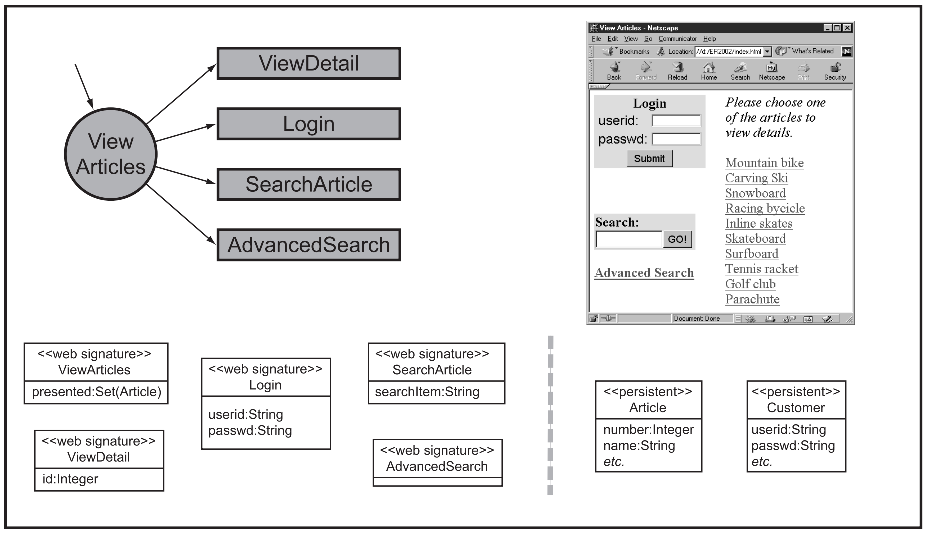

Figure 2: Example analysis model

adheres to the page's signature and fulfills the output specification that is annotated at the relevant transition. In general there may be more than one transition between a server action and a client page, used to model conditional computation of different client page parameters.

A server input specification for a page/server-transition indicates that the system is not yet completely modeled with respect to the system's behavior upon violation of this constraint. Consider the webshop example from the previous subsection once more. The client page offers the user the possibility to login. The following server input constraint expresses that a username and password submitted by the user must be valid.

```
viewArticlesTOlogin
server input :
Customer->allInstances()->exists(userid = login.userid
                                  and passwd = login.passwd
                                 )
```

In a later, more elaborated version of the analysis model it must be described how the dialogue is continued on reception of an invalid username/password combination. The issue of server input constraint is addressed again in section 4.

As described so far the recommended server action specification already provides a tight description covering all functional aspects of this kind of system component. Furthermore nothing of the effort made in server action specification is overhead because all found constraints may be reused in system implementation. Beyond this, for the time being, our approach does not prescribe how to specify the data processing associated with a server action, i.e. the side effect on the system data state. Every ad hoc pseudo code notation may serve for this purpose. We recommend to refrain from describing this type of functionality by any kind of artificial pre/post-condition specification that necessarily uses some modal operator.

2.5 Enabling Conditions

An enabling condition for a page/server transition specifies, whether a submission capability is offered to the user depending on the current system state and the history of the dialogue which led to the current client dialogue state. For this purpose the new OCL property "along" is introduced which can be applied to a path expressions consisting of state names and describing a path in the form chart. The resulting expression evaluates to true if the current transition's source client page

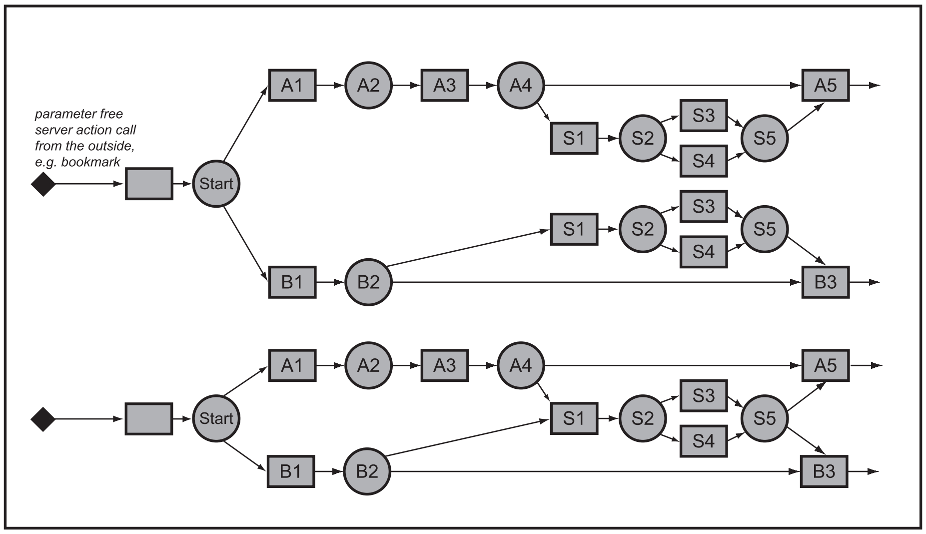

Figure 3: Modeling use cases

has been entered through states as specified in the path. This notation element makes enabling conditions a key concept for flexible and succinct modeling of even complex use cases. Though the form oriented approach to software engineering is not use-case driven, but feature driven, the use case still remains an important concept. In the form chart every path can be considered a use case if appropriate.

In the following example (Fig.3) a system is described by two semantically equivalent form charts. The system comprises to major use cases A and B. At a certain point in each of the use cases a supporting use case S may be entered which is the same in both cases. After finishing the supporting use case, the respective major use case is reentered. The first description does not use enabling conditions. Instead it makes use of the possibility that a state may occur more than once in a form chart. We don't explain the semantics of multiple state occurrence in detail here. In the alternative second description the following enabling conditions are used.

```
s5TOa5
enabled:  s3.s2.s1.a4->along() or s4.s2.s1.a4->along()

s5TOb3
enabled:  s3.s2.s1.b2->along() or s4.s2.s1.b2->along()
```

Each of the description styles has its advantages because there are tradeoffs concerning global and local complexity and understandability with respect to the whole diagram and a sole diagram state. A simple instance of the above example is found in webshops. The customer can enter the ordering use case in nearly every situation. After finishing the ordering the user wants to be offered an explicit link to the dialogue state from which she has once entered the ordering, i.e. she does not want to be forced to use the browser's history mechanism for this purpose.

There may be more than one transition between the same client page and server action. These transitions must carry explicit distinguishing labels. If all of the several transitions are completely unconstraint this simply amounts to redundancy which may actually be found in today's business to customer systems. The need of several transitions is obvious with respect to enabling conditions and client output specification.

2.6 Discussion

Every state is a unit with an input signature. Dependent on an actual parameter and possibly additional system state this unit produces output of different signatures, i.e. we deal with units having data dependent types, theoretically understood in the type system community [3] and FP community [1] as term dependent types. Though theoretically challenging, there is nothing mysterious about specifying such units during analysis, on the contrary, it is highly natural. Note that the bubbles in the very successful Structured Analysis approach are data dependent typed units, too, though this may not have been explicitly stated in the past.

3 Structuring the Form Chart

3.1 Tree-like Feature Decomposition

Every sub graph of the form chart is called a feature chart. Two feature charts are consistent if they are equal in all overlapping parts. Two feature charts are combined by graph union. A form chart decomposition is a collection of consistent feature charts, so that the combination of the feature charts yields the complete form chart. A feature chart can be decomposed further the same way. The result is a tree of chart decompositions. Decomposition makes the form chart manageable, it is the result of organizing the form chart artifact during the analysis phase. Beyond this the tree structure has no predefined semantics; every combination of feature charts, even from different levels of the tree, yield a correct sub graph of the form chart. Beside this rather strict top-down view of structuring the model, the notion of feature gains importance during form storyboarding, a proposed so called feature-driven requirement eliciting approach outlined in section 5.

Appropriate balancing rules for this decomposition approach can be defined and supported by tools.

3.2 Menu-like User Interface Parts

Another notation element is the client page set that is depicted by a double lined bubble. It is annotated by a list of page names and serves as shorthand notation for these pages. A feature chart may be annotated as menu. Then, if the feature chart contains a client page set all transitions starting from there must be deleted from all charts in the current decomposition. Affected states must reference the respective menu feature chart by an explicitly given name. Fig.4 shows how the described mechanism fosters readability of system interfaces with menu-like user interface parts.

4 Refinement

Form oriented analysis allows for different grades of detail. It does not require complete specification. We do not confine form oriented analysis to a single process model in this paper. Instead we provide a set of well defined incomplete abstraction layers called refinement stages. Refinement itself is defined by using the notion of feature: the model B is a refinement of model A iff A is a feature of B. In an informal sense form charts can be seen as refinements of form storyboards. We now explain the most important refinement stages:

4.1 Signature Model

This model contains the complete data model and the complete form chart, but no constraint annotations in both diagrams. This model is valuable as the bare metal model giving the complete structure of the user interface and the data.

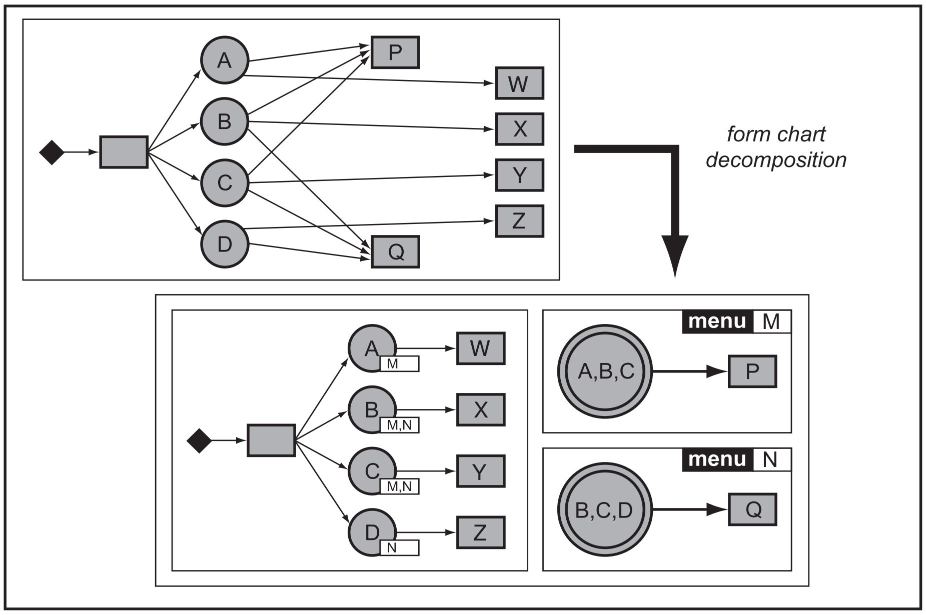

Figure 4: Modeling menu-like user interface parts

4.2 Server Input Declared Model and Server Input Safe Model

Server input constraints have been explained as being related to user input that does not meet the requirements, e.g. the user enters a sum above its limit. Server input constraints have to be replaced in later stages by branches from the server page for these cases. These branches are of minor interest, therefore it is helpful if their full specification can be deferred. A server input declared model is the model that contains server input constraints. The server input safe model is the model where all server input constraints are replaced by branches leaving the server state.

4.3 Multi-User Declared Model and Multi-User Safe Model

Several exceptional cases are caused by the submit/response based use of multi-user systems and therefore called multi-user exceptions. These exceptions are due to the fact that submit/response based systems are typically based on an optimistic business logic approach. An example are systems with shopping carts. A typical strategy is that the items in the shopping cart are not reserved for the customer. It is assumed a rare event that the item has been sold if the customer buys the item. This is the optimistic assumption. The multi-user exception occurs whenever the optimistic assumption fails. The multi-user declared model is the model in which all multi-user exceptions are excluded by server input specifications. The multi-user safe model is the model in which these constraints are replaced by branches leaving the server state.

5 Form Storyboarding

Form oriented analysis is supported by the requirements engineering technique of form storyboarding [6]. Form storyboarding is a set of activities for requirements eliciting. The resulting documents of the requirements engineering activities are form storyboards. Form storyboarding is supposed to take place before analysis. Form storyboards are similar in structure to form charts. They are bipartite state transition diagrams with client pages and server actions, but are principally ambiguous and informal in well defined aspects. Hence they combine formality and informality in such a way that the best possible support of discussions with domain experts is enabled.

Form storyboards should contain only textual descriptions of constraints. They do not have a separate data model beside the state transition diagram. Instead the signature of input forms is written directly into the server action.

Form storyboards are amenable for feature driven composition as well as for refinement. For short we note that it is discouraged to use form storyboards for a transaction safe refinement if the application has been agreed upon to be optimistic, which is the common case.

A special coarse grained version of form storyboards is the page diagram, in which no input form signatures are written in the server states. The page diagram is commonly used as the diagram in which the pictorial representation of the client page is included into the client page symbols. Hence the page diagram corresponds to the concept of a nonexecutable GUI prototype in other approaches, but it is still embedded into the refinement hierarchy of form storyboards. A requirements gathering approach for use cases which yields diagrams similar to page diagrams, but with different semantics, are User Interaction Diagrams [22].

6 Advanced Topics

In the following we give an outline of advanced concepts within the form oriented analysis method.

6.1 Multi-Window Analysis

The system interface can be used with several pages visible in different windows, as it is possible with standard web browsers. The user can spawn new windows, but this should not be necessary for meningful system usage. Multi window interaction is an additional option for the user for more convenient system usage. Hence the challenge for the analysis process resulting from user caused multi window interaction is only to make the system safe with respect to window spawning.

6.2 Accessing External Interfaces

In many projects a part of the task of the new system may be to access external services. Examples are legacy systems and web services. External interfaces have turned out to fit well into the form oriented analysis pattern. The modelling of external interfaces within form oriented analysis maps with standard solutions for external interfaces [20] [13]. Form oriented analysis covers ingoing, outgoing as well as bidirectional external interfaces.

6.3 Further Topics

User roles are handled in the same way as in form storyboarding. Session parts which must be handled securely can be specified by a single feature which contains all secure dialogue parts and is annotated as secure. A textual format for form charts comparable to the GENTLY language [4] is currently under construction.

7 Web Based System Architecture

Form oriented analysis leads to a crucial mitigation of the impedance mismatch between analysis and design. Consider the "redirecting request application model" [14] proposal to web system architecture, coined model 2 architecture, for example. Recall that many state-of-the-art architectural frameworks rely on the model 2 architecture [17] [19]. There is a canonical mapping of a form chart to a model 2 architecture. Every server action is realized as front component that is responsible to parse the transmitted parameter stream and to process the necessary business logic according to the dialogue constraints of the form chart. Every client page is realized as presentation component. Beans are used to transmit output generated by a server action to client pages.

Form oriented analysis fits seamlessly to an own proposal for architecture of web based information systems [7] that improves current approaches because it is based on a new statically strongly complex typed server pages technology [5].

8 Related work

Structured Analysis [12] 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 [23] 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 [10] to state-of-the-art versions [11] of this approach the recommended human computer interface specification techniques exclusively target the modeling of GUIs.

State diagrams has been used for a long time in user interface specification [18], partly with the objective of user interface generation [16]. All of these approaches target graphical user interface specification only, at a fine grained level concerning user interaction within one page. Another early approach [9] targeted modeling of push-based, form-based systems, i.e. single-user desktop databases. All these approaches are far from being full fledged analysis approaches.

9 Conclusion

Form oriented analysis is a modeling technique that is designed from scratch to fit the special needs of engineering a certain ubiquitous kind of scalable commercial systems that has been characterized as form based, submit/response style systems. Form oriented analysis possesses the following characteristics:

- Form oriented analysis is an elaborated approach that from the outset takes into consideration advanced topics concerning both the modeling activities and artifacts like model decomposition and refinement and the driving forces of the solution domain like e.g. transactionality and scalability.
- Form oriented analysis gives precise semantics to OCL-annotated transition diagrams for modeling the user interface while unifying forms and links as strongly typed method calls. It defines decomposition and refinement concepts.
- Form oriented analysis comes along with form storyboarding, a requirement engineering method for eliciting functional requirements and communicating them with domain experts.
- The relationship between form oriented analysis models and state-of-the-art proposals for software architecture of web based information systems is well understood.

References

[1] L. Augustsson, "Cayenne - a language with dependent types", Proceedings of the International Conference on Functional Programming '98, ACM SIGPLAN Notices, vol. 34, no. 1, Jan. 1999, pp. 239-50.

[2] C. Brabrand, A. Møller, M. Ricky, M.I. Schwartzbach, "Powerforms: Declarative client-side form field validation", World Wide Web Journal, 3(4), 2000.

[3] N.G. de Bruijn. "The mathematical language AUTOMATH, its usage and some of its extensions". In Symposium on automatic demonstration, Lecture Notes in Mathematics 125, Springer, 1970, pp. 29-61.

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

[5] D. Draheim, G. Weber, "Strongly complex typed dialogue-oriented Server Pages", Technical Report No B 02-05, Intitute of Computer Science, Free University Berlin, March 2002, http://www.inf.fu-berlin.de/inst/pubs/tr-b-02-05.ps.gz

[6] D. Draheim, G. Weber, "An Introduction to Form Storyboarding", Technical Report No B 02-06, Intitute of Computer Science, Free University Berlin, March 2002, http://www.inf.fu-berlin.de/inst/pubs/tr-b-02-06.ps.gz

[7] D. Draheim, G. Weber, "An Overview of state-of-the-art Architectures for Active Web Sites", Technical Report No B 02-07, Intitute of Computer Science, Free University Berlin, March 2002, http://www.inf.fu-berlin.de/inst/pubs/tr-b-02-07.ps.gz

[8] M. Gogolla, M. Richters, "On Constraints and Queries in UML", The Unified Modeling Language - Technical Aspects and Applications, Physica-Verlag, Heidelberg, 1998, pages 109–121.

[9] 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.

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

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

[12] T. DeMarco, "Structured Analysis and System Specification", Prentice-Hall, 1979

[13] OBJECT MANAGEMENT GROUP, "The common object request broker: Architecture and specification", Tech. Rep. Version 2.0, Object Management Group, July 1995.

[14] E. Pelegri-Llopart, L. Cable, "Java Server Pages Specification, v.1.1." Sun Press, 1999.

[15] M.Richters, M.Gogolla, "On Formalizing the UML Object Constraint Language OCL", Proc. 17th Conf. Conceptual Modeling (ER98), Springer LNCS, 1998

[16] 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.

[17] Malcolm, D. Struts, an open-source MVC implementation. In: IBM developerWorks, February 2001.

[18] 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.

[19] Webmacro. http://www.webmacro.org/, 2002.

[20] Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl.

[21] J. Warmer, A. Kleppe, "The Object Constraint Language", Addison Wesley, 1999

[22] P. Vilain, D. Schwabe, C.S. de Souza, "Modeling Interactions and Navigation in Web Applications", Proceedings of 7th International Workshop on Design, Specification and Verification of Interactive Systems, LNCS Vol.1921, Springer-Verlag, Salt Lake City, Utah, October 2000, pp. 115-127.

[23] E.Yourdon, "Modern Structured Analysis", Yourdon Press, Prentice-Hall, 1989