EXPLOITING UML EXTENSIBILITY IN THE DESIGN OF WEB INFORMATION SYSTEMS

Ekaterina Gorshkova
eVelopers Corporation
cathy@evelopers.com

Boris Novikov*
University of St. Petersburg, Russia
borisnov@acm.org

Abstract The application design and development methodologies based on UML are well-known, widely used in practice, and proved to be critically important for object-oriented design. However, these methodologies and tools do not provide specific support for modeling issues peculiar to web applications.

This paper defines an UML extension capable to refine the design of the client part of web application. Several new diagrams are specified with provide for precise definition of the content of web pages, navigation between them, and their relationships to business data.

Keywords: UML, Web application design, Rapid application development, Navigation, Browsing semantics

1. Introduction and Background

A design and development of a Web application differ significantly from more traditional stand-alone and client-server ones. It usually has to comply with several kinds of requirements like scalability, security (which are common for client-server and web), and portability of the client software (which is more specific for web applications). They are supposed to provide dynamic generation of the user interface, advanced presentation features, and be easily customizable.

*The work of the second author was partially supported by Russian Foundation for Basic Research under grant No 01-01-00935.

One of the main advantages of the web application is that it uses web browser as client. This client is ultra-thin; almost no configuration is required on the client side and the deployment procedure becomes simpler. However, the server part of web application typically has a complex distributed architecture. The system architect usually has to fight with n-tiers, each with its own set of services, protocols and APIs.

All these requirements and architectural circumstances make the design of such an application a challenging task, even when high-quality advanced modeling methodologies and tools are used.

Several methodologies and tools supporting them exist for modeling, design, and development of web applications [Conallen, 1999, Isakowitz et al., 1995, Ceri et al., 2000, Fraternali and Paolini, 2000] for both server parts and client interfaces. However, widely accepted methodologies designated to support application logic design, e.g. UML-based, do not support directly modeling of the client side, especially user interfaces. This paper intends to fill in the gap between application logic and client interface modeling.

Our paper concentrates on client aspects of web application. We provide the UML-based(UML) [Booch et al., 1998] framework for modeling the content of web pages, navigation between them, and their relationships to business data. This framework enables a possibility to use same or compatible tools for all parts of the application and provides an opportunity for accelerating web application development.

1.1 Overview of the Proposed Approach

As Web system is a hypermedia system, we have to model what pages we have, how they are linked together and what data they show. We do not assume that application logic is provided by certain technologies. If web pages are dynamic, they can be produced by CGI scripts, J2EE technologies, Active Server Pages or something else. We only imply that the conceptual or implementation data model of the application is created. And we show the relationships between the web pages and the data they display.

We do not stick to any particulary software development process. Different processes iterate development of the web application through the more or less similar steps [Conallen, 1999, Ceri et al., 2000, Garzotto et al., 1993, Isakowitz et al., 1995]. We assume that design of our system passes the following phases:

1 Requirements gathering

2 Design of the server part

3 Hypertext design

4 Design of the content of web pages

Here we consider the third and the fourth phases. The notation used in this work is an extension for Unified Modeling Language (UML) [Booch et al., 1998]. Our approach describes the system as a set of diagrams including:

- Conceptual model of the application
- Navigational diagrams
- Composition diagrams

This methodology helps to provide more robust and consistent design of the application. The tool that supports these diagrams can provide forward engineering of the web application as well as simply generating the prototype of the user interface. The opportunity to separate the functional model of the user interface from presentation features is important. These two diagrams can be given to UI designer as requirements. Another important opportunity is presenting the relationship between user interface and the data it shows. This relationship can be used for checking model consistency in the early stage and for generating the code that validated input data.

1.2 Example

We illustrate the techniques proposed in this paper with an example of simplified online banking application. The user of this application can make electronic payments and money transfers, monitor and manage accounts, view and search transactions on specific account. The application must ensure discretional security for both users and the enterprise.

There are some additional requirements and refinements:

- The system requests additional authentication prior to acceptance of any withdrawals from user's account.
- To perform search on transactions the user should be able to specify various search criteria such as time period, range for the transferred amount, or payee name.
- New payment orders may be created as copies of existing transactions with subsequent editing.
- The user should have access to the online help system.

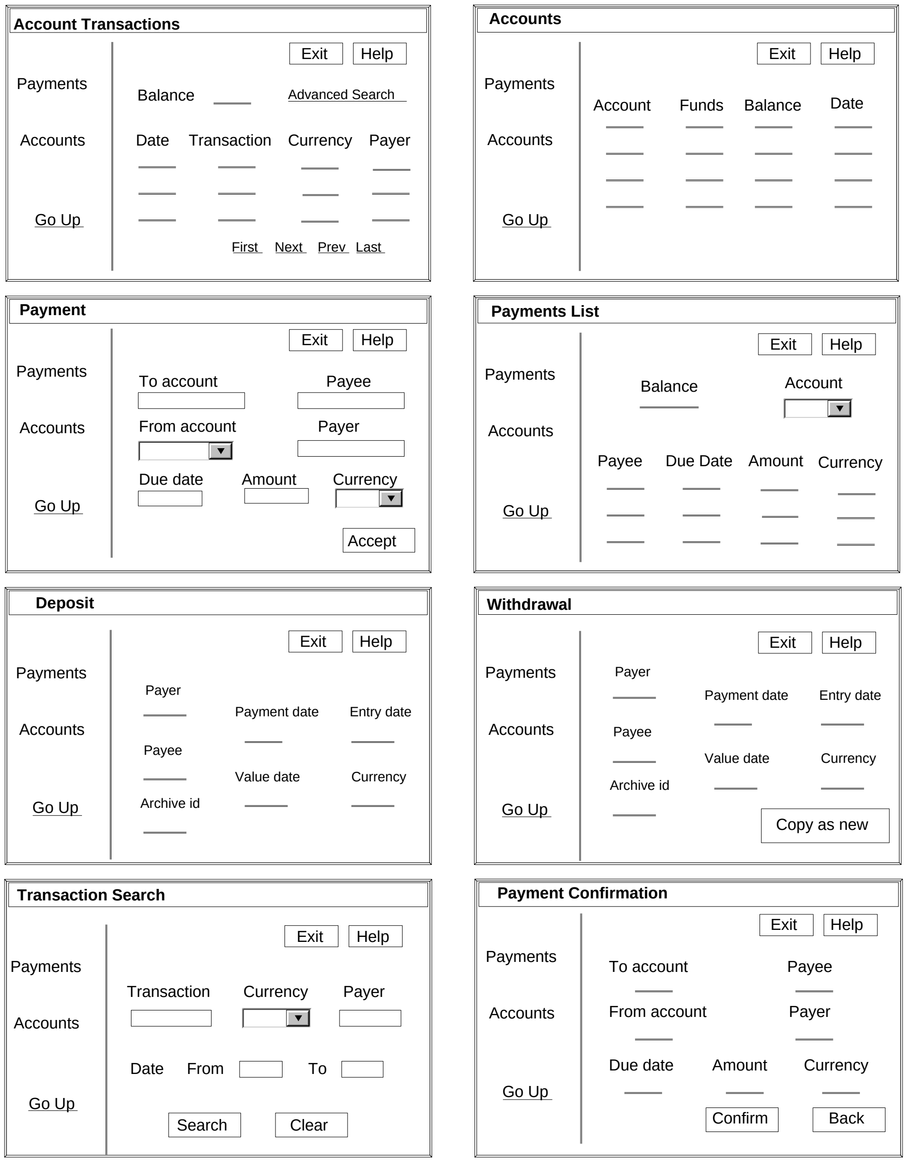

Figure 1. Bank Application Example.

In the subsequent sections we explain how these requirements can be addressed at the design phase. The final result of the client part design is shown in the figure 1.

1.3 UML Extension Mechanisms

UML provides a rich set of modeling concepts. However, when modeling applications in special domains, designers often want to extend the language by adding new elements or attaching new semantics to existing elements. UML gives possibility to extend the standard constructs without modifying the meta-model. These extension mechanisms are designed so that tools can use them without understanding their full semantics. The new building blocks are expressed in terms of stereotypes, constraints and tagged values.

- Stereotype can be viewed as an existing meta-model element with new semantics. Instances of the stereotype have the same structure as the base element, though they may require additional constraints and properties. For the tool this new element behaves as the old one. It can also be rendered with a special icon.

- Constraint specifies the condition for the model to be well formed.It can be attached to the arbitrary element. The specification does not define the interpretation language of the constraint. It can be natural human language, mathematical formula or one of the programming languages. We use Object Constraint Language (OCL) - a formal language for expressing constraints on object models.

- Tagged Value is an extension of the element's property. Every model element has information attached to it in terms of tag-value pairs. Stereotypes always use tagged values to store additional properties not supported by base element. The values of these properties can be constrained.

2. Related Work

After two decades of research and development in the area of application modelling and design, several methodologies converged into few widely accepted ones, the most important of them are supported by Unified Modelling Language UML [Booch et al., 1998]. However, these general methodologies do not address specific aspects of Web applications.

Several research efforts addressed hypermedia application design focused on data-intensive web-sites [Atzeni et al., 1998, Fernandez et al., 1998, Fraternali and Paolini, 2000, Garzotto et al., 1993, Isakowitz et al., 1995]. All these proposals provide their own design methods. For data model they use entity-relationship model or its extension, for navigation they provide predefined navigation constructs. These methods are accompanied with diverse notations and tools, while our approach is based on a single language.

Another recent and the closest to us approach is WebML language [Ceri et al., 2000], which provides orthogonal models for designing structure, composition and presentation. However, it uses its own restricted notation and fails to express advanced composition and navigational constructs. We avoid this kind of limitations relying on the power of UML.

The choice of the standard modeling language is important. Firstly UML becomes a de facto standard for application modeling. Secondly it can be easily extended for specific domains and at last it is currently supported by many tools.

Modeling of abstract user interface for Web is covered in [M. Abrams, 1999]. It does not provide any visual notation, but uses XML for expressing all user interface concepts.

Several authors applied statecharts for modeling hypertext to present behavioral and structural features [Turine et al., 1997, Zheng and Pong, 1992]. These approaches are excellent for hypertext behavior, but they fail to model complex structure of the static elements in the web pages. We avoid this problem by separating navigational and composition issues into different diagrams.

Modeling of web applications with UML is described in [Conallen, 1999]. This work covers various aspects of the topic. The web application extension for UML suggested by this work allows to model separately client and server aspects of the application. The client part of this notation enables to express web pages structure, its relationship to various technologies(ActiveX, Java, JavaScript, etc.) and page navigation. However, this approach uses only class, i. e. static, diagrams to represent both behavioral and structural things, reducing the clarity. Another point is that the stereotypes chosen for this notation are bound to the certain technologies, complicating the creation of conceptual abstract design.

3. Modeling Presentation Features

From the presentation layer perspective, a web application is just a collection of interlinked web pages with dynamically provided content. From the client side the Web application is seen as a linked collection of Web pages. In this paper we identify and address two issues related to the description of the presentation layer of the web application:

- The content of each particular page which might be displayed to the user, namely, which data sources from other layers contribute to the content and how these data are placed on the page.

- The structure of inter-page links and their relationship with certain actions in the application.

To address these modeling issues, we introduce navigation and composition diagrams. The former concentrates on modeling user behavior expressed in terms of events and transitions, while the latter models page content and represents page structure.

When modeling the client side of the application we do not pay attention how information is structured in the server part of the application, but how it is presented to the user. However, we can use thick client pattern. This pattern allows the client to execute some logic. The most widely used technology is JavaScript. To perform more complicated tasks on the client side Java applets or ActiveX components can be used. These elements are related to the implementation rather then conceptual model, and we do not consider them here.

3.1 Navigation Diagram

The navigational diagram is implemented as a special case of the UML statechart diagram. It is intended to model navigation issues of the hypertext and helps to visualize the behavior of the user in the system. The whole web application is modeled as state machine attached to the abstraction of the user. When browsing the Web application the user comes to various pages. Each of the pages can be modeled as state and browsing as transition between states. The user leaves the state when some event occurs. This event is initiated by user (when he clicks the hyperlink or presses submit button) or by the system (when he is redirected to some page after a certain amount of time). Pages are presented in this diagram regardless of their content. We are interested only in the following:

- The page that is presented to the user at a given time.

- The event that calls a transition to the next page.

- The transition itself, i.e. where the user goes when some event occurs.

An example of the navigation diagram for the bank application is shown on figure 2.

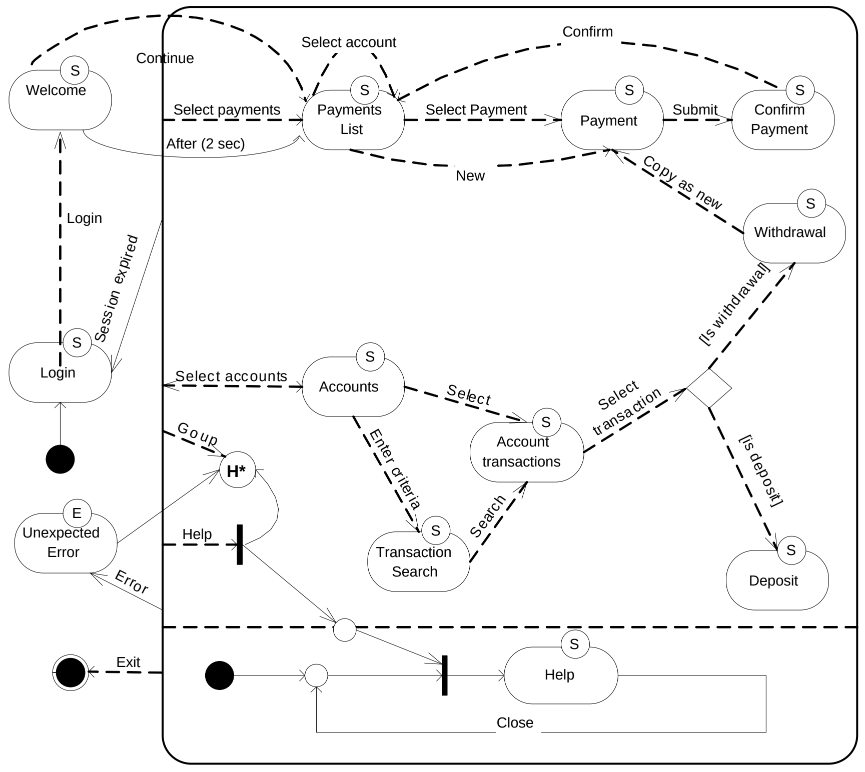

Figure 2. Example of the Navigation diagram.

This diagram brings the client-side view at glance. It shows the linked collections of pages presented to the user, the alternatives of the browsing and exceptional situations. The next subsections describe in detail the elements of this diagram.

3.1.1 Screens and Transitions. The central notion we need for this diagram is a screen. It captures an html page that is presented to the user in the browser window. Each window corresponds to exactly one page, regardless of the way it was composed. Each screen in this diagram corresponds to a state. To emphasize its special semantics we create the state stereotype <<`screen`>> and assign to it a special icon - rectangle with rounded corners and letter S in the circle in the upper right corner.

Transition is a user movement between states. The user moves from one page to another when some event occurs, either user-initiated or system. The user initiates an event by a clicks the link or presses a button.

A system event may be exemplified by time-out. We distinguish these types of events by assigning the special stereotypes to the transitions that they trigger. Transitions initiated by user request are stereotyped as <<`user`>> and are shown with dashed line. Transitions initiated by the system are stereotyped <<`system`>>. They do not have special notation. An example of different transitions is shown on the figure 2. In this diagram we have two outcoming transitions from the welcome screen. If the user presses Continue link he goes to the Payment List screen, otherwise he is redirected to this screen automatically after two seconds.

A state in UML may have internal transitions, i. e. transitions that do not change active state. In principle, these transitions might be used for modeling of certain portions of client side application logic. However, we do not recommend to use these modeling techniques in navigational diagrams as it refers to composition issues.

Another situation is with self-transitions, i.e. transitions that forces to leave the state and then to enter it again. We self-transition is used to show that we perform the trip to the server for some action. In the figure 2 in the Payments List screen only payments for specific account are shown. If the user changes the account the system load new list of payments from the server.

We often need to model events that the user can initiate on every screen. These may be moving backwards, returning to the home page, launching the help screen or performing some other common actions. This situation is modeled by creating composite state with high-level transitions. The typical presentation solution of such problem is a frameset. In the frameset we always have navigational pane in the top or left side of the page. When clicking on the links in the navigational page the target panes are changed.

This means that only one of the target panes can be active at the moment. So, these target panes can be modeled as sequential substates of the composite state. We can move from any target pane to any other target pane, so the transitions to them are made from the boundary of the composite state, allowing all the nested states to inherit them. Such modeling technique does not assume any presentation details. The only thing we are interested in here where we can go from the current state and what event triggers the transition. This page can be implemented as either a frameset, button menu or collection of links included in every page.

The figure 2 shows three high level transitions that represent links from navigation pane: Go up, Payments and Accounts. The user can initiate two more events on every page: exit from the application and bring help screen. Triggering these events also lead to high-level transitions. These two situations are identical from the point of navigation issues.

There may be an alternative what page is shown to the user, depending on the some circumstances. This alternative is represented as a guard condition. In the figure 2 we may go from Account Transactions screen to the Deposit screen or Withdrawal, depending on the type of selected transaction.

The application usually has an entry point. Typically, it is a login screen. This entry point is modeled as initial state.There can also be a final state that is reached when the user logouts from the application.

3.1.2 Displaying several windows at once.

Sometimes we need possibility to display to the user several documents at once in different browser windows. These windows can be peers or child window can be opened as a pop up from the main one. Each window contains html page that corresponds to the notion of screen in our model. The general idea is to model multiple windows as concurrent regions of the composite state. This decision enables us to see for each moment what windows do we show to the user. This also helps to define the when the window should be opened or close avoiding lost windows.

In this situation we may have some synchronization issues. For example, popup window is typically launched after activating some elements in the main window. To synchronize behavior of these two windows we put between the regions the synch state.

In our example the Help window is shown as a popup after pressing $\boxed{\text{Help}}$ in the main window.

3.1.3 Exceptional States.

When moving from one page to another some exceptional situations may happen. There can be a critical failure in the server or the user session may expire. These states are stereotyped <<`exceptional`>>. This stereotype is a child of <<`screen`>>. An outcoming transition from this state usually goes into history state. The icon is the same as for screen, except the letter S in the circle is changed to E for "exception".

3.1.4 Well-Formedness Rules.

In order to enforce certain constraints on the stereotypes defined in this paper, we introduce several rules, in addition to the constraints imposed on all elements of their kind. Few examples of such additional rules are listed below.

Screen. A rule for Screen states that all screens should be reachable:

```
self.incoming->notEmpty
```

Exceptional. An <<`exceptional`>> state can have only <<`system`>> incoming transitions.

```
self.incoming->forAll( t |
    t.stereotype.name = "system")
```

User. The <<`User`>> transitions can happen between <<`screen`>> or composite states.

```
(self.source.oclIsKindOf(CompositeState) or
self.source.stereotype.name = "screen") and
(self.target.oclIsKindOf(CompositeState) or
self.target.stereotype.name = "screen")
```

3.2 Composition Diagram

The composition diagram is a special case of class diagram. We use it to express structure of the web pages and identify their content: how they are connected together and what data is carried from one page to another. We also show how pages are related to business data that they represent. The composition diagram provides static view of web pages, rather than behavioral features. Only links that are essential for structure are shown in this diagram and neither conditional nor exceptional situations are represented here. This diagram can be used for generating JSP, HTML and JavaScript code.

An example of such diagram is shown in the figure 3. For simplicity only three pages are modeled.

3.2.1 Pages. Navigation diagram shows the all the screens of the application and conditions of browsing. To model the content of the screen it can be mapped to one class that presents its static structure. This class is placed into composition diagram and is stereotyped <<`page`>>. All the elements of the screen like buttons, links and input fields are modeled as attributes of the corresponding class. The tool may provide links from pages to screens and vice versa to show their relationship.

To share common elements between pages and to provide more understandable structure some parts of the page can be selected into separate classes. Thus a page may play role of the container for other pages which are modeled as aggregated classes. In the figure 3 pages Account,

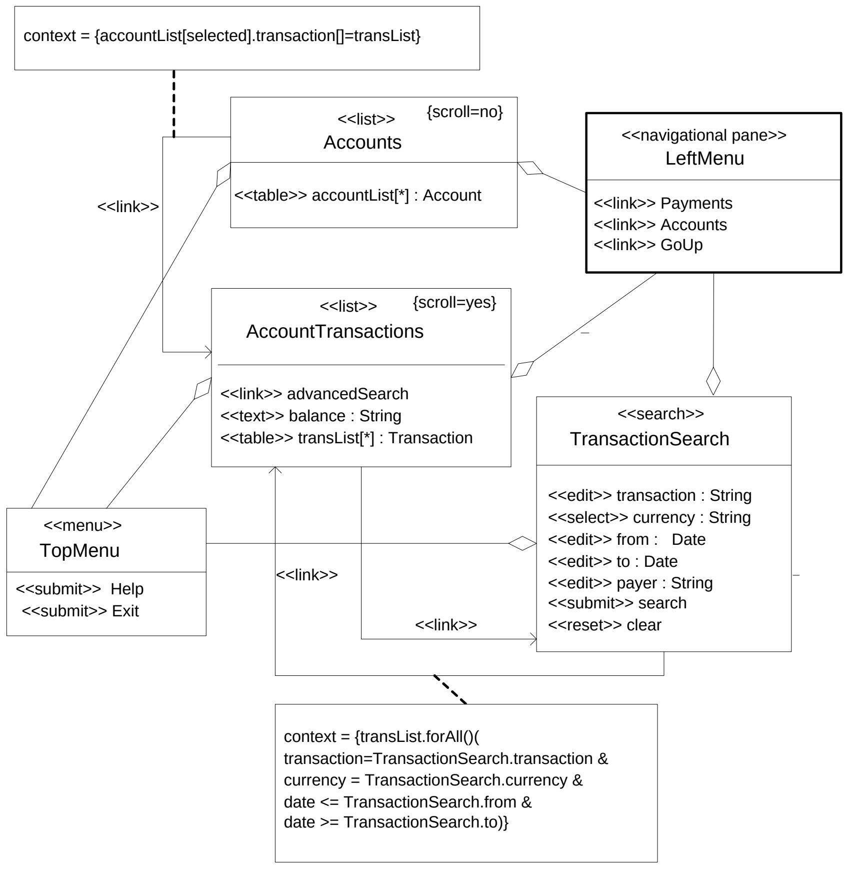
context = {accountList[selected].transaction[]=transList}
<<list>>
{scroll=no}
Accounts
<<table>> accountList[*] : Account
<<navigational pane>>
LeftMenu
<<link>> Payments
<<link>> Accounts
<<link>> GoUp
<<link>>
<<list>>
{scroll=yes}
AccountTransactions
<<link>> advancedSearch
<<text>> balance : String
<<table>> transList[*] : Transaction
<<search>>
TransactionSearch
<<edit>> transaction : String
<<select>> currency : String
<<edit>> from : Date
<<edit>> to : Date
<<edit>> payer : String
<<submit>> search
<<reset>> clear
<<menu>>
TopMenu
<<submit>> Help
<<submit>> Exit
<<link>>
<<link>>
context = {transList.forAll()(
transaction=TransactionSearch.transaction &
currency = TransactionSearch.currency &
date <= TransactionSearch.from &
date >= TransactionSearch.to)}

Figure 3. Example of the Composition diagram.

AccountTransactions and TransactionSearch contain LeftMenu and Top-Menu as nested pages.

The page has elements

3.2.2 Forms. Any form is a page, usually with controls that expects some user input. It is modeled as a class with stereotype <<`form`>>. This stereotype is a child of <<`page`>>. The semantics of the form is a container of input controls, each of them contains the value that will be passed to the server on submitting. The input can be taken using various widgets like select, multiselect, edit, radio button and checkbox. Classes for these control can be defined in the separate package of the model or some standard hierarchy like DOM classes can be used. These controls are the attributes of the <<`form`>> class.

We also define a special stereotype for a search form - a form that allows the user to type some values in editing fields that are used for selecting a set of objects with matched values. This kind of form is modeled as class stereotyped <<`search`>> that is child of <<`form`>>. In the figure 3 the page TransactionSearch is used for specify criteria for searching transactions.

Each value that is passed to the server is somehow reflected in the business model. Typically, it is assigned to some attribute of the business objects. The relationship between the business object and form that is used for its editing should be shown. For these purposes we introduce binary navigable association <<`view`>> from control to business object attribute. Notation for this association is shown in the page TransactionSearch in the figure 3. It is indicated as short arrow placed on the left side of the control attribute pointing to the name of the attribute of business object. These associations can be used for generating code (client or server side) that performs validation and parsing the http request.

3.2.3 Collections. The application may have pages that show a list of entities. Typically it is a result of certain query. For this kind of pages we suggest <<`list`>> stereotype that is a child of page. Classes of this stereotype are required to have an attribute of some collection type stereotyped <<`table`>> that contains data to show in this page.

The <<`list`>> page contains tag *scroll* that should be equal to *yes* or *no*. The value *yes* indicates that the list is shown by portions and *next*, *previous*, *first* and *last* are available. In the figure 3 we have two <<`list`>> pages AccountTransactions and Accounts.

3.2.4 Links. After defining pages we need to connect pages to make hypertext. Usually the user jumps to other page when he clicks the hyperlink or presses submit button. This behavior is reflected in the navigational diagram. However, we need to show what information is carried from the source page to target one. The navigable association between source and target pages is stereotyped <<`link`>>. Each <<`link`>> has a tag *context* with expression as value. The expression determines what information to show in the target page depending on the state of the source page. The expression can be written in the human language, pseudocode or OCL. If *context* is empty, then pages are connected in a free way and no information is sent from source page to target one. In the figure 3 *context* of the link shows how the data on the page AccountTransactions is derived from the data on the TransactionSearch page.

3.2.5 Well-Formedness Rules.

Link. <<`Links`>> association must be put between pages :

```
self.allConnections->forAll( ae |
    ae.type.stereotype.name = 'page')
```

List. Any <<`List`>> must have at least one attribute stereotyped <<`table`>>:

```
self.feature.select->(a | a.oclIsKindOf(Attribute)
    and a.stereotype.name = 'table')
```

4. Conclusions and Future Work

In this paper we introduce a UML extension to capture specific aspects of Web application design, especially modeling of the client side.

The model consists of a set of diagrams describing navigation and composition of dynamic web pages exposed to the end user. The resulting diagrams can be used for building HTML prototype of the application and for generating client(HTML,JavaScript) and server side code. We are planning to refine the application design by modeling information that can be personalized.

The major advantage of this approach is that it provides for uniform description of all application components and overcomes a gap between design of application logic and presentation features, which are usually done with isolated tools.

In the future we are going to improve the methodology of the web application development and provide detailed description of the development process not only for the design phase.

We are also planning to develop a set of object constraints which may be used to describe relationships and dependencies between different diagrams and enable verification of the application at early design and development phases.

The relationship between business objects with their constraints and presentation layer can be used for generating test data set, that can be uploaded into automatic test system.

References

[Atzeni et al., 1998] Atzeni, P., Mecca, G., and Merialdo, P. (1998). Design and Maintenance of Data-Intensive Web Sites. In *Proc. EDBT'98*, Lect. Notes in Comp. Sci. v. 1377, pages 436–450. Springeer-Verlag.

[Booch et al., 1998] Booch, G., Jacobson, I., and Rumbaugh, J. (1998). *The Unified Modeling Language User Guide.* Addison-Wesley.

[Ceri et al., 2000] Ceri, S., Fraternali, P., and Bongio, A. (2000). Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33(1-6):137–157.

[Conallen, 1999] Conallen, J. (1999). Modeling web application architectures with UML. *Comm. ACM*, 42(10).

[Fernandez et al., 1998] Fernandez, M. F., Florescu, D., Kang, J., Levy, A. Y., and Suciu, D. (1998). Catching the Boat with Strudel: Experiences with a Web-Site Management System. In *Proc. ACM SIGMOD Intnl. Conf.*, pages 414–425.

[Fraternali and Paolini, 2000] Fraternali, P. and Paolini, P. (2000). Model-driven development of web applications: the autoweb system. *ACM TOIS*, 18(4):323–382.

[Garzotto et al., 1993] Garzotto, F., Paolini, P., and Schwabe, D. (1993). HDM - A model-based approach to hypertext application design. *ACM TOIS*, 11(1):1–26.

[Isakowitz et al., 1995] Isakowitz, T., Stohr, E., and Balasubramanian, P. (1995). RMM: A methodology for structured hypermedia design. *Comm. ACM*, 38(8):34–44.

[M. Abrams, 1999] M. Abrams, C. P. e. a. (1999). UIML: an appliance-independent XML user interface language. In *Proc. WWW8, Elsevier*, pages 617–630.

[Turine et al., 1997] Turine, M. A. S., de Oliveira, M. C. F., and Masiero, P. C. (1997). A navigation-oriented hypertext model based on statecharts. *Hypertext*, (1):102–111.

[Zheng and Pong, 1992] Zheng, Y. and Pong, M. (1992). Using statecharts to model hypertexts. In Lucarella, D., editor, *Proc. ECHT92*, pages 242–250, Milan, Italy.