

Filling the Gap: New Models for Systematic Page-based Web Application Development & Maintenance

Carlos F. Enguix and Joseph G. Davis

Decision Systems Laboratory
Department of Business Systems, The University of Wollongong
Northfields Ave. Wollongong NSW 2522, Australia

cfe01@uow.edu.au, joseph_davis@uow.edu.au

Abstract

The current state of application development on the WWW is characterised by anarchy and ad hoc methodologies. In recent years, various hypermedia methodologies have been proposed to facilitate the deployment of Web applications. However, no standard methodology has emerged to cater the needs for a systematic and methodological approach to complex and dynamic Web application development. This paper presents the design and specification of a multi-paradigm model suited for representing the interaction in a client-server environment of procedural, declarative, object-based, event-oriented and OO components, suited especially for page-based Web applications.

Keywords: Page-based Web Application Development, Multi-paradigm Models, Implementation Models, Client-server Interaction, Web Software Engineering

1. Introduction

The current state of application development on the WWW is characterised by anarchy and ad hoc methodologies. Despite the increasing popularity of Web-based application production systems, no standard methodology has emerged to cater the needs for a systematic and methodological approach to Web application development. In recent years various hypermedia methodologies and CASE tools such as OOHDM (Schwabe and Rossi 1998), W3DT (Bichler and Nusser 1996), AutoWeb (Paolini and Fraternali 1998) have been proposed to facilitate the adaptation of the hypermedia world to the Web. However, it has been observed that there is a clear impedance mismatch between the models represented in the hypermedia world at the conceptual, navigational and presentation levels of abstraction and the models required for complex and dynamic Web applications with procedural, declarative, event-oriented, OO and client-server capabilities. The design of a multi-paradigm model suited for representing the interaction in a client-server environment of procedural, declarative, object-based, event-oriented and OO components in page-based (HTML-based) Web applications are proposed in this paper.

The paper is organised as follows: in section 2 we explain the factors that motivated us to define a new framework and a new model specifically adapted for Web application development. In section 3 is presented an overview of some of the most important Web-based software engineering models that are related to our research. A multi-paradigm model suited for page-based-based Web applications is presented in detail in section 4. Finally we conclude the paper with conclusions and directions for future research.

2. Motivation

At the end of the autumn session of 1998 we were finalising a research project which involved the design and implementation of domain-specific object-relational database search engines (Enguix et al. 1998). The prototype consisted of a demonstration of the construction of a schema-based object-relational database search engine which focused on the Australian Universities domain. The architecture of the database search engine is entirely based in the tight integration of logical and physical subsystems that interact, collaborate

and complement each other. Due to the complexity involved some kind of model was badly needed to depict in an intuitive how client and server components interact and collaborate among each other. As a matter of fact, we needed a graphical model capable of representing and integrating:

- Hypermedia behaviour inherited from HTML constructs
- Software engineering models and methodologies needed to depict either procedural, event-oriented, object-based or object-oriented behaviour
- Identification, definition and distribution of client/server components

We investigated many hypermedia design models such as OOHDM (Schwabe and Rossi 1998), RMM (Isakowitz et al.1995), W3DT (Bichler and Nusser 1996) and so on. We found out that structured hypermedia design methodologies do not include implementation models in their final life cycle stages as implementation is implicitly viewed generally in all of them as largely ad hoc. As we know, traditional software engineering life-cycles do not incorporate hypermedia models.

This situation led us to work on a new multi-paradigm model that addresses the requirements mentioned above, capable of presenting in an intuitive way how client-server Web applications can be developed in a modular fashion. The model must support several software engineering paradigms that coexist, collaborate and integrate to form a new complex hybrid environment which we may refer to as the “Web application paradigm”. The richness of a Web application is such that it might require the interaction of declarative, procedural, object-based and pure OO components that are executed in an imperative, declarative or event-oriented environment.

An IDC white paper entitled as “Web Application Development Perspectives” McClure (1998) presents a detailed survey of the current state of Web application development. This study attempted to provide a forecast of emerging trends. It reported the following findings:

- **General coding approach:** the coding approach for the majority (85 %) of Web applications is a combination of HTML pages with embedded scripts (page-based Web applications)
- **Client-side components:** the most important features required in Web applications were the use of HTML, the use of JavaScript® and Java™ components in client Web browsers
- **Current server-side components:** a majority of Web developers deployed Web applications with Cold Fusion Markup Language (CFML), server-side JavaScript®, JAVA™, PERL and C/C++ and the CGI.

This survey also underscored the lack of an intuitive, adaptive prototyping framework for the systematic design, development, reuse and maintenance of Web applications. This paper addresses this problem and focus on the design and specification of multi-paradigm graphical models especially suited for page-based Web applications that typically interact with back-end databases.

3. Related Work

Some distinct approaches have influenced the current state-of-the-art in the Web application development arena. On the one hand, the hypermedia world had a considerable impact on the design and implementation of static page-based Web applications. On the other hand, software engineering principles and models have been critical in the deployment of complex and dynamic Web applications. In reality, a Web application may integrate seamlessly both hypermedia and software engineering principles and models. In this section we present an overview of some of the more interesting Web-based information systems and software engineering models suited for Web application design.

The WWW Design Technique (Bichler and Nusser 1996) is a design technique which enables to model both structured and unstructured Web sites, supporting both static and dynamic components. The W3DT methodology includes an intuitive graphical model suited for the deployment of Web sites which describes design primitives such as sites, page, index, form, menu, link, dynamic link, etc.

The WebComposition model (Gellerson 1998) represents Web entities as component objects with a state and behaviour associated with each entity. Components can model Web entities of any arbitrary granularity such

as individual links, a complete Web page, a Perl Script and so on. The WebComposition models file-based resources into OO components which enables the reuse and maintenance of Web components but does not address modelling the actual behaviour of Web applications in run-time environments.

An extension of UML for Web application design is presented in (Conallen 1998). A Web application is considered as a specialised version of a client/server application due to the fact that connections between client and server sides exist only during a page request. A set of new classes and association stereotypes are created to model in an object-oriented way Web applications. Because a Web page may contain logical components that are executed either on the client or the server side it can be represented in the model as two different classes, one belonging to the stereotype class server page and the other to the client page. Variables that are page scoped are modelled as attributes of the class and internal modules as methods. Stereotyped classes specify hyperlinks (link), component availability (server and client component), represent specialised Web pages (forms, framesets), destination of a Web page (target) and so on.

4. The WebApp Framework

Currently Web applications are characterised by their complexity inherited from the integration and interaction of client/server components/subsystems and by the integration of hypermedia and software engineering principles. As cited previously the WebApp framework proposes a multi-paradigm environment that may include procedural, declarative, object-based, OO, event-driven components and so on. It also proposes a formal or semi-formal multi-methodological approach that permits the combination of software engineering and hypermedia models. Additionally, we consider as a critical necessity the design of an intuitive model and suitable notations capable of representing the interaction of page-based client-server Web applications. The WebApp framework addresses all these requirements in order to facilitate the systematic design, implementation and maintenance of page-based Web applications.

Web applications typically call for a fine-grained prototyping framework. As a result, the extent of inter-dependencies and interactions between the different phases and iterations, the prototyping life cycle for such applications is quite complex. In our WebApp framework we propose a six-stage prescriptive life cycle model for Web application development: requirements gathering, simulation modelling, architecture modelling, implementation scenario analysis, formal modelling, implementation modelling. The last stage involves the actual implementation of the application. High-level implementation details are shown with the aid of two graphical models especially suited for page-based Web applications that serve as a basis for the final implementation:

- **WebApp TLNC:** WebApp Top-Level Navigational Charts
- **WebApp ALSC:** WebApp Application-Level Structure Charts

As research in progress, we are trying to formalise all the phases involved in the WebApp framework life-cycle. Our focus in this paper is, however, on the models obtained in the implementation phase. From our experience, the models obtained in this phase facilitate the deployment and maintenance of Web applications. Furthermore, we consider as a promising research direction the integration of these models into Web Application IDEs and CASE tools.

4.1 WebApp TLNC: WebApp Top-Level Navigational Charts

WebApp TLNCs are used to show from a high-level perspective the hypermedia structure and organisation of page-based Web applications. A WebApp TLNC shows in a compact and efficient way the hierarchies of Web pages and the possible navigations between these in a Web application. The symbols represented in TLNC are explained in detail in the following section 4.2.

Figure [fig. 2](#) shows an example of a WebApp TLNC representing a Web application used in the development of the object-relational search engine. The purpose of the application is to query entities associated to the University domain, such as Faculty home pages, Staff member home pages, etc. The main_query application is frames-based and allows the selection of entities to be queried on the left hand-side frame via a parameterised link. On the right frame is loaded a dynamically generated form which allows us to query the

selected entity when submitted, calling recursively to the same Web page that contains the form and appending retrieved results to the end of the Web page. The retrieved results may allow us to navigate to the Web pages associated to each entity retrieved or to a Web page associated to entities that are related to the retrieved entity. Note that there is a clear distinction between a call/link process and a load process. This feature allows us to specify in a simple and intuitive way the load of WebApp components in different target WebApp containers (i.e. form_query is called from the left-frame of the main_query WebApp Web page and loaded on the right-frame)

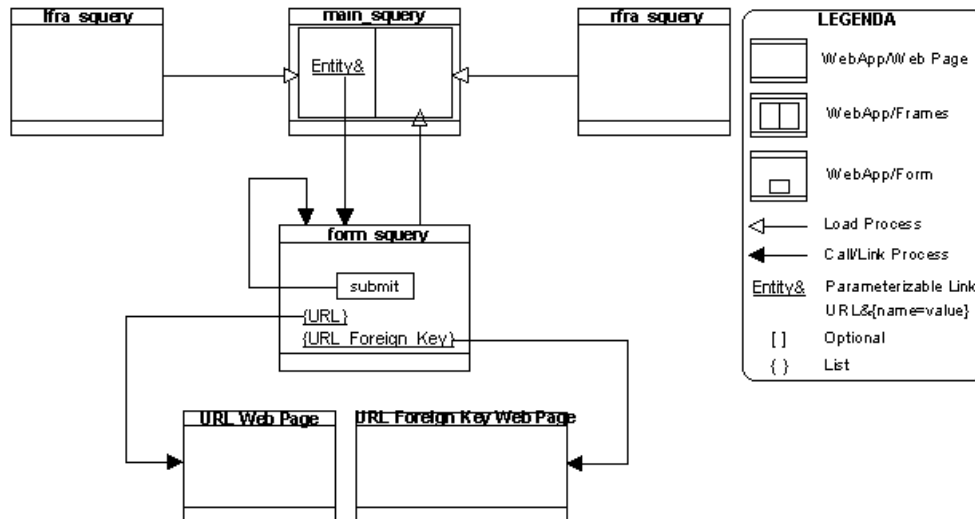


Figure 2. Example of a WebApp TLNC

A TLNC does not provide much implementation details. The purpose of TLNC is to serve as a basis for further refinement. Application-level details such as parameter passing, client-server interaction and so on is shown in WebApp Application-Level Structure Charts (WebApp ALSC).

4.2 WebApp ALSC: WebApp Application-Level Structure Charts

WebApp ALSC are similar in concept to procedural structure charts, in that the modularity of the application and parameter passing is shown. In addition navigational paths can be derived from HTML form submissions or following links. Taking into account that Web applications may contain OO, object-based, declarative, procedural, event-oriented components either in the client or the server side, ALSC charts can be adapted in order to represent in an intuitive way a multi-paradigm environment. In addition, we distinguish three types of charts that facilitate the isolation of client and server sides and that depict the interaction between both:

- **Server-side:** the focus is on the server-side as the majority of processes are executed on the server-side. Client components are excluded from this chart showing in a very high-level of detail server-side components. Server-side components may include page-based declarative Web applications (Informix Web Data Director App Page Tags, Cold Fusion Markup Language), object-based/OO server-side scripts (Server JavaScript®, Microsoft ASP), traditional CGI scripts, JAVA™ Servlets, dynamically loadable executable modules (3/4 GL generated module) and so on. Server-side components normally interact with relational/object-relational/OO DBMS or other legacy databases or may interact with distributed modules/objects via DCOM, CORBA, RMI or RPC.
- **Client-side:** the focus is on the client-side, characterised by being event-oriented. The majority of modules are internal to a Web application and are activated on the Web browser. This chart must show all client-activated components and the interaction between them. Client-side components generally include client-activated object-based scripts (JavaScript®, VBScript) and OO objects (JAVA applets, ActiveX components). Client-side modules/objects augment the efficacy of server-side applications, delegating some server functionality to client Web browsers.
- **Client-Server interaction:** both client and server sides are included, showing only top-level client-based and server-side components. The components that are formed by integrating/collaborating both sides are

marked as C/S (Client/Server). The parts that are activated on the server-side are marked as S and the parts that are triggered on the client-side are identified as C.

A WebApp ALSC includes the following elements and the corresponding symbols, depicted in table 1:



















Symbol	Notation	Symbol	Notation	Symbol	Notation
	Web Page		Call/link process		Explode symbol
	Frame		Load process		Data Parameter
	Form		Generate process		Control Parameter
	Applet/ActiveX	<u>Link</u>	Link		Activation point
	Sub-component list/set	<u>Plink&</u>	Parameterised link	Ei	Event
	Virtual Web Page		Client-side Activated Component	Ci	Condition
	Internal Active Component		Server-side Activated Component	{ }	List/Set of Elements/Objects
	External Active Component		Client-server side Activated Component	[]	Optional Element/Object

Table 1. List of elements/symbols included in WebApp Application-level Structure Charts

- **WebApp Web Page:** represents a Web page component which is a container of other sub-components
- **WebApp Frame:** a container of other page-based components (Web page, Form). A Frameset may contain an arbitrary number of frames. By default we represent a frameset with two frames associated
- **WebApp Form:** represents a single application Web page with embedded forms. By default we represent one form per Web page.
- **WebApp Applet/ActiveX:** represents a Web page component which contains embedded JAVA applets or ActiveX components
- **Sub-component list/set:** includes a set of object instances which are normally generated dynamically. May include rows of data, HTML table and lists, links, etc.
- **Virtual Web Page:** virtual unnamed Web page generated dynamically. May include Web pages generated by CGI scripts, Servlets, procedural HTML blocks, etc.
- **Internal Active Component:** internal subroutine, module or object embedded in Web applications. Generally represents client-activated objects such as JavaScripts, JAVA applets and ActiveX components.
- **External Active Component:** external subroutine, module or object called by a Web application. Generally represents server-activated objects such as Server-side JavaScripts, CGI modules, DLLs, Servlets, etc.
- **Load process:** indicates the non-assisted automatic load of components via declarative statements. Used especially in WebApp frame-components to indicate in which frame a Web page is loaded.
- **Call/link process:** indicates automatic load of components via reaction of active components or via end-user interaction. May include calls to internal and external active components (subroutines, modules, objects) or active components called by form submissions or by following links/parameterised links or clicking on any JAVA-based/ActiveX GUI object that submits data to the server

- **Generate process:** normally references to HTML components that are generated dynamically (i.e. tables, lists, error pages, URL indexes, etc.)
- **Link:** link to a URL. In order to provide flexibility and semantics in the notation, we can define any link stereotype contextual to the current application being deployed (i.e. URL, URL Foreign key, index link, etc.)
- **Parameterised link:** a link that includes parameters of the form of name/value pairs (i.e. URL?variable_name=variable_value). In order to provide flexibility and semantics in the notation, we can define any parameterised link stereotype contextual to the current application being deployed. To distinguish from plain links the stereotype name should terminate with an '&' symbol (i.e. Entity&, Table&, etc.)
- **Client-side Activated Component:** includes active components that are active on the client-side and HTML-based components that are entirely generated on the client-side.
- **Server-side Activated Component:** includes active components that are active on the server-side and HTML-based components that are entirely generated on the server-side.
- **Client-server side Activated Component:** includes active components, HTML-based components (Web page, Form, Frame, sub-component list/set, etc.) or entire Web applications that are active or are formed by interacting client and server sides
- **Explode symbol:** represents either a module/subroutine that explodes into a more detailed WebApp ALSC or a high-level application object that explodes into a detailed OO class diagram that may include state and behaviour or separate state transition diagrams (STD)
- **Data Parameter:** primitive data type, constant, vector or object passed to/generated by an active component (subroutine, module or object), a form submission or a parameterised link with the aim of passing data between components
- **Control Parameter:** primitive data type, constant, vector or object passed to/generated by an active component (subroutine, module or object), a form submission or a parameterised link with the aim of passing state information between components
- **Activation point:** either represents a condition or an event that activates a given active component (subroutine, module or object) either procedurally or event-oriented
- **Condition:** represents an evaluation of the state of control parameters or the values stored in data parameters or both
- **Event:** includes end-user generated GUI events (i.e. onSubmit form), non-assisted automated GUI events (i.e. onLoad form) and any other kind of event such as rules-based and so on.

One of the most desirable features provided by WebApp ALSC charts is the capability of representing in a simple and elegant way a multi-paradigm environment. Activation points may involve an imperative/object-based/OO generated procedural condition or an object-based/OO generated event. An explosion point may lead to a detailed WebApp ALSC or to a detailed OO class diagram.

The list of elements and symbols included in table 2 not only permits us to represent the majority of current page-based Web applications that typically interact with back-end databases in a modular and intuitive way but facilitates their maintenance as well. The distinction between internal and external active components is critical for maintenance purposes. Internal components are embedded within Web pages whereas external components may represent external dynamic link libraries, distributed objects and so on.

Due to the fact that the HTTP protocol is a stateless protocol where connections between client and server are open during a single operation and that HTML source code is interpreted sequentially from top to bottom, we can distinguish the following most important events associated to client-server interaction in page-based Web applications:

- **OnLoad:** represents the initial load of a Web page, which may be triggered by previous form submission, the selection of a link/parameterised link and so on. The event is depicted in the ALSC as an Activation point.
- **OnReload:** represents a recursive call to a given Web page which may be triggered by a form submission, the selection of a parameterised link, etc. The event is represented in the ALSC as an Activation point.
- **OnSubmit:** affects to form submissions. Data and/or control parameters are passed to the server-side. The event is represented implicitly in the ALSC as a Call/link process and may include an Activation point (i.e. data reformatting previous to form submission).
- **OnClick:** generally affects links/parameterised links although we may include buttons or any GUI object associated to JAVA™ applets that transmit data to the server. In this case the event is represented implicitly in a WebApp ALSC as a Call/link process.

This simple event model permits us to represent the interaction between client and server components on a majority of page-based Web applications.

4.2.1 Client-Server Interaction WebApp ALSC

The complexity of developing client-server Web applications partly resides in demarcating and identifying when the server-side or client-side must be activated and how the two sides must collaborate. In general terms, there are two kinds of collaboration between client and server side in typical Web applications:

- **From server to client side:** data is received from the server and processed a-posteriori by the client. Examples may include dynamic generation of form objects, generation of tables or lists of row instances retrieved from a DBMS, dynamic generation of URL links, etc.
- **From client to server side:** data is pre-processed on the client-side and sent to the server-side. Examples may include validation and reformatting of data previous to form submissions.

Client-server interaction WebApp ALSC concentrate on the collaboration between the client and server sides, therefore only top-level active modules/objects are shown. Any of these top-level modules/objects may explode into more detailed client or server WebApp ALSC.

Figure [fig. 3](#) shows an example of the final Client-Server interaction WebApp ALSC representing the main_query application. When the form is submitted query values are tested and reformatted a priori (E2). A successful form submission will trigger a recursive call on the form_query form component passing the query parameters (form objects + hidden fields). The query parameters are evaluated (C1, C2) activating server-side active components (C dynamic link libraries) that pre-process query parameters and execute SQL3 queries returning dynamically generated table elements with data (C4, C5). At the same time the sub-component list/set data is being generated (i.e. rows of data captured from a DBMS), a client active component (JavaScript® module) retrieves the data, captures URL columns, reformats them and converts them into hyperlinks (C3). If an error occurs while executing the query or retrieving data (C6) a dynamically generated virtual page showing error information is displayed on the client Web browser. Parameterised links (Next&, Prev&) that enable the retrieval of rows of data in groups of an arbitrary number can be generated dynamically as well acting as a sliding window (i.e. Prev 25 Next 25).

5. Future Research Directions

The models proposed in this paper have been used in the development of Web applications with client JavaScripts® and ILLUSTRATE™ Web DataBlade applications currently known as Informix Web Data Director. Having investigated a range of different technologies and tools for Web application development (IDM 1999) we contend that the majority of these can benefit from the models proposed in the WebApp framework. With respect to the automation of the design process, the idea is not to develop an isolated upper CASE tool but to try to provide add-ons to popular CASE tools such as Rational Rose, Platinum Paradigm Plus, Oracle Designer/2000, etc.. Another alternative is to provide add-ons to popular Web application development IDEs & tools such as Oracle WebDB, NetDynamics, Cold Fusion, NetObjects, Informix Web Data Director, etc.

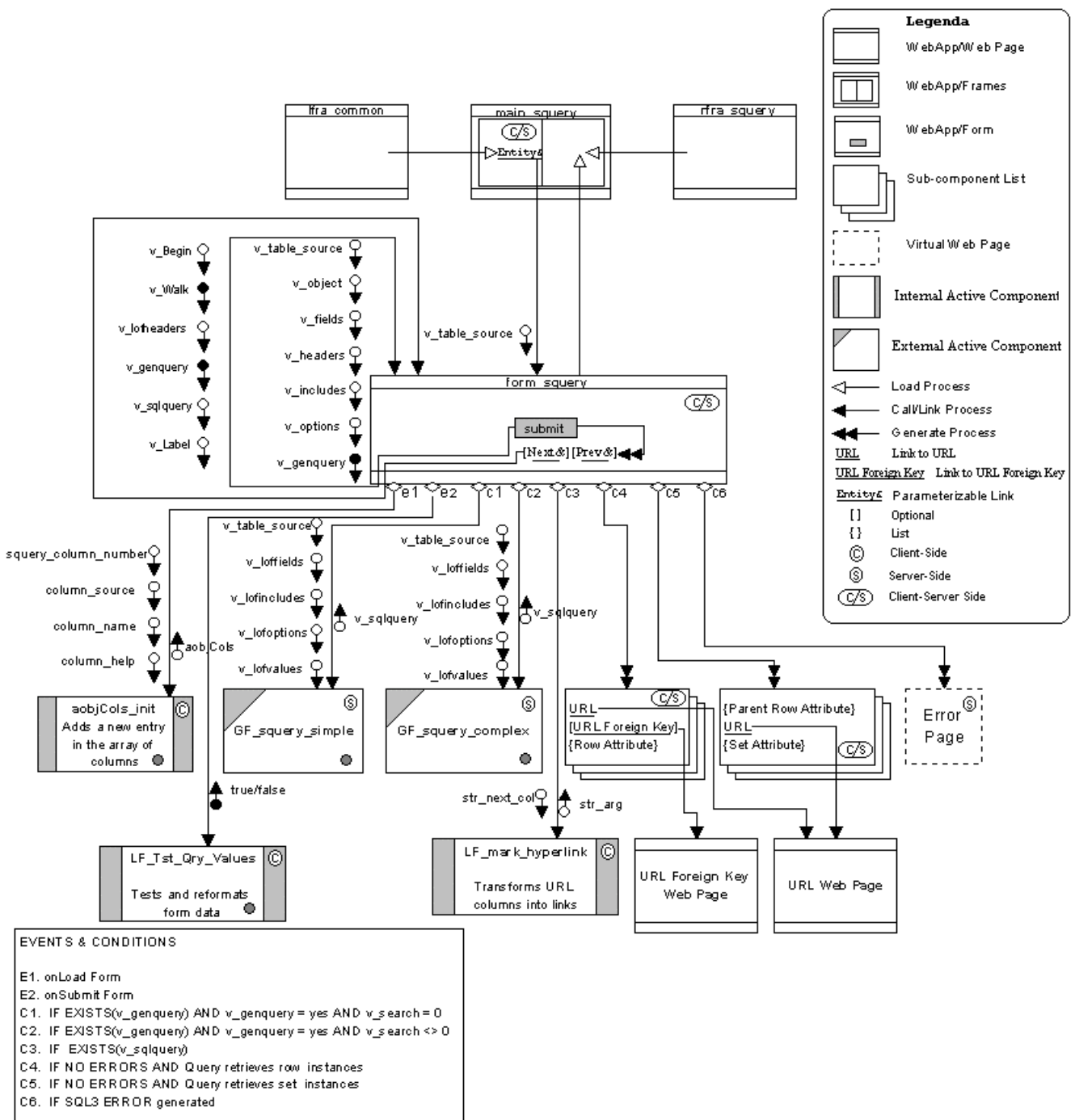


Figure 3. Example of a Client-Server interaction WebApp ALSC

6. Conclusions

Two modelling notations are presented, WebApp Top-Level Navigational Charts and WebApp Application-Level Structure Charts used to facilitate the systematic design and maintenance of complex page-based Web applications. The former is used to provide a high-level logical map of the structure of page-based Web applications. The latter supports a multi-paradigm notation and is used to show implementation details and run-time behaviour, especially client-server interaction between software components.

This framework is likely to be of benefit to Web developers for a systematic Web application development and maintenance process. It also fosters the reusability and optimisation of software components. Other communities that may benefit from the research proposed in this article are Web designers, analyst/programmers, project leaders/managers and end-users which may have the chance to have a better understanding of the systems being deployed. This approach is sufficiently general to cater to the needs of a wide-cross section of page-based Web application environments, tools and languages.

7. References

(Bichler and Nusser 1996)

Bichler, M. and Nusser, S., "Developing Structured WWW-Sites with W3DT", Proceedings WebNet 96, San Francisco, California, USA, October 1996

<http://aace.virginia.edu/aace/conf/webnet/html/223.htm>

(Conallen 1998)

Conallen, J., "Modeling Web Application Design with UML", White Paper Conallen Inc, June 1998

<http://www.conallen.com/ModelingWebApplications.htm>

(Enguix et al. 1998)

Enguix, C. F., Davis, J.G., Ghose, A.K., "Database Querying on the World Wide Web" Decision Systems Laboratory, Technical Report TR98/5/101, May 1998.

http://budhi/uow.edu/postgrad/carlos/tr98_5_101.htm

(Gellersen 1998)

Gellersen, Hans-Werner "Object-Oriented Web Engineering", First International Workshop on Web Engineering (WebE 98) WWW7 Conference, Brisbane Australia, 14 April 1998.

<http://www.teco.uni-karlsruhe.de/~hwg/webe.html>

(IDM 1999)

IDM, "Web Application Development Tools", Intranet Design Magazine, March 1999

<http://idm.internet.com/tools.shtml#Head>

(Isakowitz et al. 1995)

Isakowitz, T., Stohr, E.A., Balasubramanian, P., "RMM: A Methodology for Structured Hypermedia Design", Communications of the ACM Vol. 38 No.8, August 1995

(McClure 1998)

McClure, S., "Web Application Development Developer Perspectives", An International Data Corporation (IDC) White Paper, November 1998.

<http://www.allaire.com/documents/reports/IDCReport.html>

(Paolini and Fraternali 1998)

Paolini, P. and Fraternali, P., "A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications", Proceedings EDBT98 Conference, Valencia, Spain, March 1998.

<http://www.ing.unico.it/autoweb/Papers/autoweb2.zip>

(Schwabe and Rossi 1998)

Schwabe, D. and Rossi, G., "An Object Oriented Approach to Web-Based Application Design", Draft Paper Current Status OOHDM, Departamento Informatica, PUC-RIO, Brazil July 1998

<http://www.inf.puc-rio.br/~schwabe/papers/OOWebAplDesign.pdf.gz>