An Integrated Visual Framework for the Human-Web Interface

Kang Zhang
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688, USA
kzhang@utdallas.edu

Mao Lin Huang
School of Computer Science
University of Technology, Sydney
NSW 2007, Australia
maolin@socs.uts.edu.au

Kei-Chun Li
Dept. of Info. and Applied Tech.
Hong Kong Institute of Education
Hong Kong, China
danielli@ied.edu.hk

[Abstract] *The design of Web sites has been largely ad hoc, with little concern about the effectiveness of navigation and maintenance. This paper presents a general framework with a human-Web interface that supports Web design through visual programming and reverse Web engineering through visualization. The paper describes the framework in the context of a Web tool, known as HWIT, which has been developed for a pilot study.*

[Keywords] *Human-Web Interface, Web visualization, visual programming, graph grammar, information filtering*

1. Introduction

The development of Web sites with complex interconnections of large number of Web pages so far has been largely an ad hoc process. There has been no commonly accepted methodology, which supports ease of design, navigation, and maintenance of sophisticated Web sites. As the number of Web sites is increasing in an exponential order, with the huge information space provided by the Web, users become increasingly confused when they navigate a growing number of Web sites; finding the right information also takes longer time. The problems are partially due to the unstructured nature of the current organization of Web sites. For example, in most of the existing Web browsers, the process of jumping from one location to another could easily confuse the user. The main reason for this is that the user does not know the current context of space with respect to the overall information space.

Attempts have been made to develop tools and facilities to support Web site construction, although most of these tools are designed only for one stage of Web design, navigation, and maintenance.

Designing a *good* Web page is considerably easier nowadays. There are many guidelines describing in details the so-called good design of Web pages [6]. Siegel [22] classifies three generations of sites ranging from default backgrounds with wall-to-wall text used in the first generation through the second generation of visual treatments such as menus and Web maps. The third generation of Web sites allows users to pursue paths designed for their needs and interests. We are now in the third generation and moving towards a more personalized multimedia capable WWW. Hinton [13] further discusses how an organization could maintain and design its Web resources with such paths. More and more organizations are embracing the idea of personalized Web site for different types of users [18]. Personalized paths designed for different individuals would enable an organization to tailor its priority and services for the individuals according to their values to the organization.

A variety of tools, including navigational tools such as browsers and lenses [14][19], history lists [9], bookmarks and filters [23], have been developed to assist users to overcome the problem of finding information in the unstructured Web space. WebOFDAV [14] also tries to help the user to visually navigate the Web by displaying a sequence of small visual frames corresponding to user's focuses of attention. Yet, these approaches do not solve the navigation problems through structured Web design and integration of Web design and navigation.

In order to improve the design and navigation of WWW, a well-designed Web structure is expected. The development of better tools that enforce structure in the design phase, while supporting fully integrated maintenance and navigation capabilities, are urgently needed. We believe that a complicated Web system can be made more structured and navigated more easily through graphical visualization and graphical interactions. More importantly, maintaining a uniform view throughout the design, navigation, and maintenance cycle can reduce considerable development effort and enhance the navigation efficiency. The goal of the work reported in this paper is to propose an integrated view throughout the Web development cycle. The major advantages of our approach are the following:

- A visual approach to constructing and navigating Web sites is easier to comprehend than the textual form. A novice user without any programming experience would find the visual approach intuitive if the visual representation could reflect one's mental image of a Web structure.
- Automatically generated by a visual language generator, the graphical Web construction and navigation tool can be rapidly prototyped and

enhanced to the end-user's needs. The generated tool is a syntax-directed visual editor that is capable of syntactic checking of any constructed Web graph. The Web site design and navigation share the same graph formalism so that the user's mental map is preserved.

- A Web site can be maintained using a site visualization tool that shows the site in the same graphical format as in the design stage. The full integration of the design tool with a Web site visualization tool also allows a user to construct new Web sites through reverse engineering based on some existing site structures and contents.

2. Related Work

To aid Web navigation and maintenance with a sense of orientation, researchers have proposed "site mapping" methods for constructing a structured geometrical map for one Web site [18]. However, they can only guide the user through a very limited region of the WWW. Other approaches define the entire WWW as a graph and then navigate the graph [1][15]. Yet, these systems do not support the integration of Web site design, navigation, and maintenance.

Various tools and methodologies have been developed or proposed in the past few years, as listed below. Most of the tools assist, in one way or another, different areas of WWW development, mainly aiming at improving navigation. For example, Fisheye-View Graphical Browser [19] adopts fisheye view filtering strategies [21] to allow logical management of documents with nested compositions. This browser degrades dramatically as the number of nodes increases.

WebMap [7] shows a 2D graphical relationship between pages. Small circles depict pages whereas links are coloured to indicate the status of destination documents. Users can visualize the document space without having to visit all documents since WebMap implements an exploratory approach to gather the documents as a batch job. However, the whole process is time-consuming and resource intensive.

PadPrints [3] is a zooming Web browser within a multi-scale graphical environment. It displays multiple pages at a time and a large zoomable information surface depicts the links between the pages. The current page is clearly shown as it is larger than other pages. The system only enhances information browsing among different documents.

WebML [4] uses a model-based approach to Web site development. In WebML, a structural model expresses the site's data content using commonly accepted modeling languages such as UML; hypertext model describes the contents and structure of the site's pages; presentation model dictates how the pages are presented with a layout specification; finally, personalization model allows group-based or individual-based content categorization.

Other tools use software engineering methodology to approach Web development. For example, WOOM [5][16] and XWMF [12] use object-based formal metadata model for designing Web structures expressed as directed acyclic graphs (DAGs). The emphasis is on the high level design for interoperable exchange and reasoning about the Web data. The OO-H (Object-Oriented Hypermedia) method uses UML-like conceptual modeling to specify navigation and presentation features. They do not address the important issue of integrated view or offer the capability of reverse engineering.

3. The Human-Web Interface

Distinguishing from the work described in Section 2, the work discussed in this paper focuses on a uniform view of the design, maintenance, and navigation of the Web. We call such a uniform view the *Human-Web Interface* (HWI).

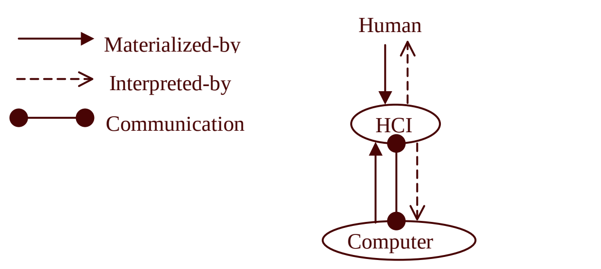
To compare HWI with traditional HCI (human-computer interface), we consider the following three aspects: the device for which the interface is suitable and designed, main functionality of the interface, and the target of the communication that the interface facilitates.

- **Device**: A HWI could be installed not only on a computer, but also on a PDA (portable digital appliance), a mobile phone, or a television set. In the latter case, the HWI needs no Web design function and thus would not be equipped with a graph editor and Web site generation engine (as described later in this paper). The display could also be much more simplified. For a PDA for example, the display may only include clickable texts and running texts for navigation and browsing, possibly with a voice interface as in WebViews [10].

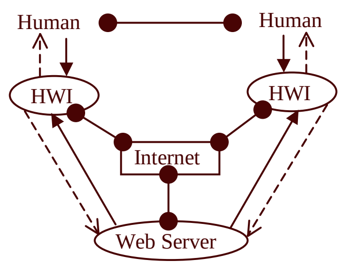
- **Functionality**: The major role of a HWI is to act as a window to the world while a HCI could just be for a standalone computer. Therefore, the main objective of a HWI is to facilitate information gathering and retrieval while that of a HCI is to facilitate operations on a computer.

- **Communication target**: Related to the above difference, the communication target of a HWI is human while that of a HCI is machine. The human-to-human communication through HWIs may be direct, such as in a Web-based net-meeting, and indirect as in usual Web browsing. To support indirect human-to-human communications in various professional domains, we need commonly understandable and agreed communication protocols. The XML standard [25] has been motivated precisely for this reason.

As illustrated in Figure 1(a), in the traditional human-computer interaction, the human user's intention is



(a) Human-Computer interaction model



(b) Human-Web interaction model

Figure 1: Human-to-computer communication in HCI and human-to-human communication in HWIs

materialized through the HCI and interpreted/executed by the computer, which in turn outputs results through the HCI to be interpreted by the user. Figure 1(b) shows the human-to-human communication model, realized indirectly through human-Web interfaces, which communicate to the Web server via the Internet. A full human-to-human communication path is described as: a human user's intention can be materialized on a HWI, which is then interpreted by the Web server according to the predefined HWI syntax and semantics (i.e. Web graph grammar); another HWI on the other end materializes and presents the user's intention according to the Web server, and the other human user interprets and understands what is presented on the HWI.

Both HCI and HWI aim at enhancing the usability and thus the user's productivity. They may therefore be developed based on similar conceptual architectures. The Model-View-Controller paradigm, or MVC for short, has been successfully used to build user interfaces in Smalltalk [17]. As one of the earliest successful object-oriented programming languages, Smalltalk supports construction of

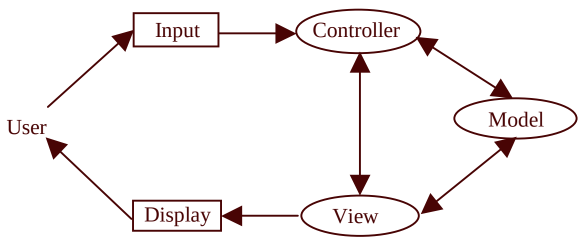


Figure 2: The Model-View-Controller paradigm

new interactive systems based on existing ones. MVC consists of three main objects as shown in Figure 2: *Model*, *View* and *Controller*. *Model* represents the application semantics, and its screen presentation is managed by *View*. *Controller* defines the way in which the user-interface reacts to user inputs.

Based on the MVC paradigm, we propose a HWI framework as shown in Figure 3, where "Graph Editor and Navigator" corresponds to *Input* in MVC, "Web Browser" corresponds to *Display*, "Filters" and "Display Markup" correspond to *View*, "Customizer" and "HWI Engine" correspond to *Controller*, and "Web Graph Grammar" and "XML Database" correspond to *Model*. The framework consists of the support for three major activities: Web site design, navigation and browsing, and maintenance and updating. The front-end of the user interface consists of a *Graph Editor and Navigator* (GEN) for Web site construction and navigation that is capable of automatic graph layout, and a Web browser that could be Netscape or Internet Explorer. This combined front-end forms the human-web interface (HWI).

The Web designer uses the *Graph Editor* of GEN to design and construct Web sites as graphs to be transformed and processed by the *HWI Engine*. The Engine performs grammatical check of the constructed graphs according to the predefined Web graph grammar [26], transforms the validated graphs into either XML documents or inter-related HTML files, and generates an internal data structure for debugging and maintenance purposes. If the generic document structure is desirable, XML document structures will be generated and stored in the XML database. The HWI engine is able to transform from one XML to another, or from an XML description to an HTML display format according to the predefined transformation grammar [28].

The HWIT framework supports several modes of displaying, including Level view, Domain view, Category view, Pattern view, and Constraint view. These views are implemented by different *filters* that are also shown in Figure 3. The Level view allows the user to choose the level of Web page pointers to display, i.e. a given level of linked

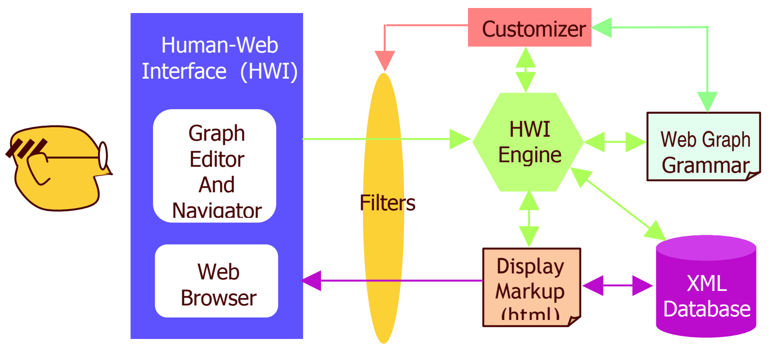


Figure 3: The HWI framework

pages in the Web graph relative to a given node. The Domain view shows the pages of a given application domain. If the Web designer has classified all the pages according to some application criteria, the user can choose Category view to see a given class of pages. The Pattern view allows the viewer to see some common patterns (as sub-graphs) in a Web graph. Finally, the Constraint view shows all the pages that satisfy a given set of constraints (e.g. a maximal file size). Filtering rules can be defined on various structures, including graph structure, Web context, and document structure [15]. Other conditions may be defined to facilitate more specialized filters.

Web designers can design or customize their own filters to suit their specific application purposes. The *Customizer* allows a Web designer or webmaster to define other desired filtering criteria, integrity conditions suited for maintenance, and syntax-directed operations associated with the Web graph grammar. For example, the user may define an integrity condition through the Customizer that no page should belong to more than one group. Web designers may also customize their designs, such as the use of graphical notations, the way in which the site will be navigated, etc, to suit the needs of domain-specific applications.

4. Using the HWI Tool

The proposed HWI framework provides a Webmaster with a uniform graphical view for the effective design and maintenance of Web sites, and allows users to navigate the Web site graphically by direct manipulation and information filtering as desired. The Webmaster designs and generates a Web site by drawing the Web graph that conceptually represents the site structure. Navigation and maintenance of Web sites are performed on the same Web graph by the user.

During design or navigation, the user can click on any graph node to enter directly into the page symbolized by the node without going through all the intermediate pages. This direct access method via a Web graph is much more efficient than linear access method in conventional browsers. The grammatical and structural organization of a Web site allows various (system or user-defined) integrity conditions for the site to be checked and any violation or inconsistency to be reported in a systematic fashion. For example, any Web pages that are orphaned by the deletion of some other pages should be detected.

We have designed an experimental tool, called *HWIT* (Human-Web Interface Tool), that realizes the above HWI functionality through visual Web programming and Web visualization. A designer or a user would be able to view any Web site from different angles, using HWIT's filtering capabilities, in a structured and personalized manner. HWIT also accepts filters that are defined and specified by users using the Customizer. The tool not only allows the user to easily navigate and explore the Web, but also assists the designer to design and maintain better-structured Web sites. Figure 4 depicts a snapshot of the HWI when navigating Kang Zhang's research home page on the Internet Explorer by a simple click on the "research" node in the graph on the navigator. The Web page shown on the IE on the right-hand side represents the "research" page in the navigation window on the left-hand side. HWIT's "Preference" dialog allows the user to choose a preferred navigational browser from various options.

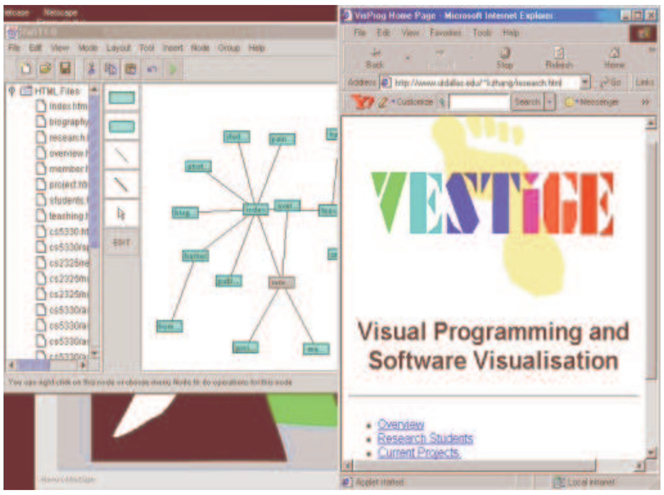


Figure 4: Navigation on a Web graph in HWIT

5. Graphical Programming for Web Design

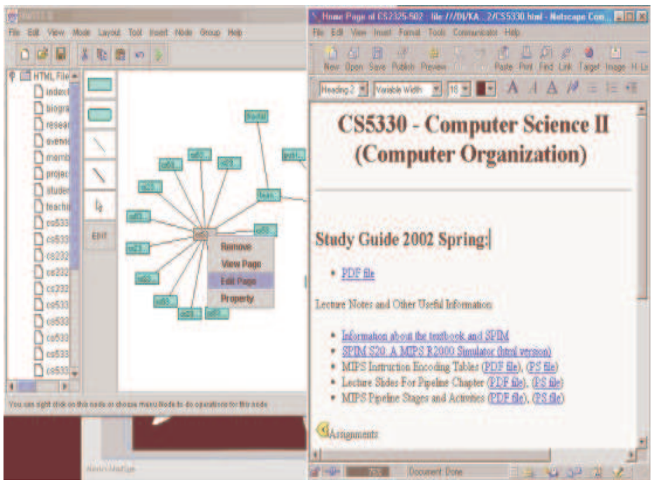
Visual structures and relationships are much easier to reason about than similar linguistically described structures. This is why designs in many application domains have been conducted on graphical representations. Using visual programming techniques to graphically design Web sites and Web pages will obviously enable more visual artists and other non-computing professionals to develop their own Web sites easily. The main philosophy behind the HWI framework is its consistent visual approach to Web design, navigation, and maintenance. This section introduces the concept of Web graphs and their notations, and describes the support for multi-versioning and reverse Web engineering through graph visualization.

5.1. Web Graphs and Design Notations

A graph G (N, E) consists of a finite set N whose members are called *nodes* and a finite set E whose members are called *edges*. An edge is an ordered pair of nodes in N. A node of a graph G1 can itself be another graph G2, which is called a *sub-graph* of G1. The properties of a graph may be inherited by its sub-graphs. We regard the organization of a Web site of any size as a graph, known as *Web graph*. A node in a Web graph represents a Web page, and an edge represents a link from one page to another. The World Wide Web is certainly the largest Web graph that is expanding all the time. For scalability and convenience of design and navigation, we define a special class of nodes, called *group*. A group represents a set of pages that are connected to a common parent page, and share the same set of attributes.

The *distance* between a pair of nodes, node A and node B, is defined as the number of intermediate nodes along the shortest path between A and B (including B). A sub-graph of graph G consisting of a node A and all such nodes in G that have a distance of N or shorter from A is called *A's level-N sub-graph of G*, or simply *A's level-N sub-graph*.

A graph class provides the general common properties that dictate whether certain operations are applicable to the corresponding graph objects. A Web graph can be constructed using a combination of tools: a graphical editor for constructing a Web graph at the high level and a Web page tool for constructing Web pages at the lower level. This is demonstrated in Figure 5, that captures a snapshot during the design of the "CS5330" home page using the Netscape Composer (launched from the "CS5330" node within the HWIT Graph Editor). The graphical editor supports two-dimensional construction of Web graphs with direct manipulation.


Home Page of CS2325-502 - file:///D|/KA...2/CS5330.html - Netscape Com...
Remove
View Page
Edit Page
Property
Document: Done

CS5330 - Computer Science II (Computer Organization)

Study Guide 2002 Spring:

- PDF file

Lecture Notes and Other Useful Information

- Information about the textbook and SPIM
- SPIM S20: A MIPS R2000 Simulator (html version)
- MIPS Instruction Encoding Tables (PDF file), (PS file)
- Lecture Slides For Pipeline Chapter (PDF file), (PS file)
- MIPS Pipeline Stages and Activities (PDF file), (PS file)

Assignments

Figure 5: HWIT Graph Editor and its connection to a Web page editor

HWIT uses a small number of simple notations, as shown on the left hand side of the screen in Figures 4 and 5, to design and visualize the components of a Web graph.

- The rectangle denotes a Node that represents a Web page. The label is used to identify the node and the page.
- The round-cornered rectangle denotes a Group, representing a group of Web pages that are combined together either due to their commonality or for the brevity of viewing. It is like a Web template or class, which can be used to generate similarly structured-pages. A Group also has a label that identifies a specific class of pages. This notation also enforces the consistency in the pages belonging to one Group.
- The thin arrow denotes an Edge that represents a Web hyperlink. This is the most common link seen in Web pages.
- The thick arrow is called a Gedge, short for Group Edge, which represents an edge coming out of or entering a Group. The difference between an Edge and a Gedge is that a Gedge connects to a Group and thus refers to all the Nodes belonging to the Group (some kind of inheritance). For example, if there is a Gedge connecting a Group A to a Node D and Node B is a member of A, then B is also connected to D. More importantly, the Nodes connected by Gedges to a Group share a common set of characteristics and attributes. This is useful in generating consistent look-and-feel pages.
- The broken-line arrow (not shown in the figures) is called a Hedge, short for Hidden Edge. It may be defined as either a connection between pages of different domains, or as a connection between a collapsed node and its neighboring node after a filtering effect. When generated automatically by HWIT a Hedge indicates the existence of a connection (in form of hyperlink) between collapsed nodes. The designer may use Hedge at the design stage. In this case, a Hedge denotes either an Edge or a Gedge between two nodes and the designer has not yet made a decision in an early stage of design.

Each constructed Web graph is syntactically verified against the Web graph grammar that is defined according to a general graph grammar formalism for diagrammatical visual languages, called *reserved graph grammars (RGG)* [26]. The main advantages of using the RGG formalism include its expressiveness and efficiency in parsing. The RGG formalism has also been used in the implementation of a toolset called VisPro, which facilitates the generation of visual languages using the Lex/Yacc approach [27].

Graphs may change over a period of time and may reach a particular state at a predetermined time. For example, a Web designer may set a time when a page or a sub-graph should be activated or disabled. A graph from an early state may be partially reused and incrementally updated for a later graph.

5.2. Support for Multi-version Web Sites

Web designers are under increasing pressure to produce updated Web sites. The conventional approach to creating and modifying a Web site is to create every single page and make changes on the copy of the source code of the page. Problems arise when the main frames of the pages are

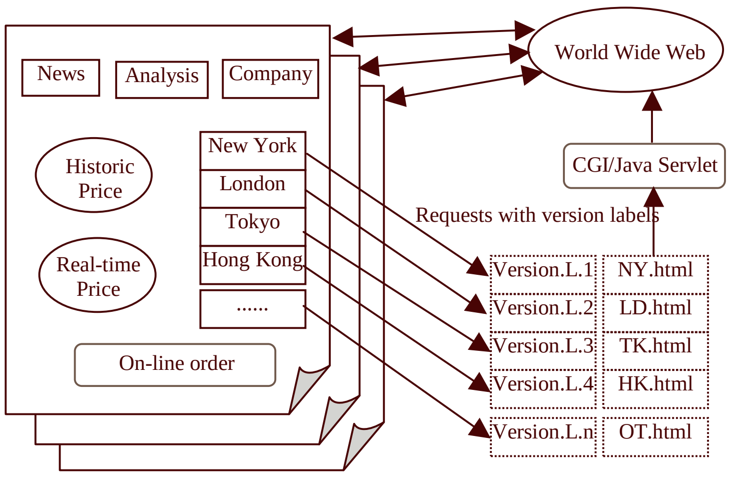


Figure 6: A multi-version Web site

almost the same while changes are needed only for part of the original documents. To allow efficient creation and modification of changing Web sites, the idea of multi-version Web sites has been proposed [24]. In a multi-version Web site, a generic source page acts as a template for other documents in the same site. The generic source page represents the common part of other documents and is used as the index page of the site. Each of the other documents can be considered a version of the site. Every request from a client is associated with a version label that is interpreted by a CGI or Java Servlet program on the Web server to point to an appropriate version of the document. This version of document is then retrieved and loaded into the template of the generic source page to be displayed. Multi-versioning is also useful when different languages or different representations are needed on a single Web site.

Figure 6 depicts a possible organization of a multi-version stock market site, in which each stock market source page (a version such as that of "New York") share a generic information page. In this case, different graph objects carry different meanings under different contexts, the semantics of a graph operation will depend on the context defined by the Web designer. Graph operations can be implemented according to their contexts but all provide the same interface to the designer.

HWIT supports the concept of associative queries for multi-version Web sites. The basic idea is that all the objects in a generic source page are categorized into three hierarchical classes: root, node and leaf, and they form a hierarchical graph. We use a data structure called *virtual version tables* (VVTs) to organize different versions of a document and facilitate the retrieval of appropriate documents.

The title page is considered the root class. Node classes include HREF links, includes, headings, and other node classes. Leaf classes are disjoint objects such as graphic files and audio files. A version label is assigned as an attribute to the root class and objects in each leaf class when submitting a request for a specific version. A hierarchically structured graph is created when the version document is generated. Information retrieval is achieved by querying the graph of the version through VVTs. More details of the associative query approach are discussed elsewhere [29].

5.3. Web Reuse Through Reverse Engineering

One of the major advantages of integrating Web design and navigation features in the HWI framework is the ability of reusing existing Web sites. A Web designer using HWIT could adapt existing Web sites by visualizing and modifying the sites, and then re-generating a new site. A Web graph under visualization is treated as a rooted tree, which consists of a set of focus nodes, each surrounded by the nodes linked to it. A focus node is usually the center of the user's attention when navigating and viewing the Web graph.

HWIT uses a force-directed algorithm [8][14] to draw existing Web sites for visualization. A *force-directed* algorithm views a graph as a system of bodies with forces acting between the bodies. The bodies are represented nodes in the graph, and the forces are relationships between the nodes in a graph and determine the geometrical positions of the nodes. A force-directed algorithm aims to compute a position for each body such that the sum of the forces applied on each body is locally minimized.

Force-directed algorithms are very popular [2]; they are easy to understand, and the results of layouts can be good. One of the most popular force-directed algorithms is called the *spring algorithm* [8]. The original spring model uses a combination of *spring* and *gravitational* forces. Edges are modeled as springs, and nodes are particles that repel each other.

To clearly distinguish the focus nodes and their neighborhoods, we have extended the spring model by adding some extra forces among the neighboring nodes surrounding the focus nodes. Suppose that $F_i=(G_i, Q_i)$ is the display frame being visualized, where $G_i=(V_i, E_i)$ is the Web graph consisting of a vertex set V_i and edge set E_i, and Q_i is the set of focus nodes. More precisely, the force on node v is:

$$f(v) = \sum_{u \in N(v)} f_{uv} + \sum_{u \in V_i} g_{uv} + \sum_{u \in Q_i} h_{uv}$$

where f_{uv} is the force exerted on v by the spring between u and v, and g_{uv} and h_{uv} are the gravitational repulsions exerted on v by one of the other nodes u in the graph. This extended spring model aims at satisfying the following three important aesthetics:

- The spring force between adjacent nodes ensures that the distance between adjacent nodes u and v is approximately equal to zero length.

- The gravitational force ensures that nodes are not too close to each other.
- The extra gravitational force aims to minimize the overlaps among the neighborhoods in the display frame and to keep the focus nodes along a straight line. More detailed description of the extended spring model can be found in an earlier paper [15].

Figures 7 and 8 demonstrate two different phases of the reverse engineering process. Figure 7 shows that a Web site is being selected for visualization, which is achieved by selecting the root page from the pop-up file management window. The layout of the Web graph is animated when the extended spring model is applied. The user may manually drag any nodes during animation to adjust and achieve user-desired layout.

After a desired layout is displayed, the user can convert the graph back to the editing mode in HWIT as shown in Figure 8. Figure 8 is a snapshot after the user has added a sub-graph at the bottom and is being editing the properties of the CS6366 node.

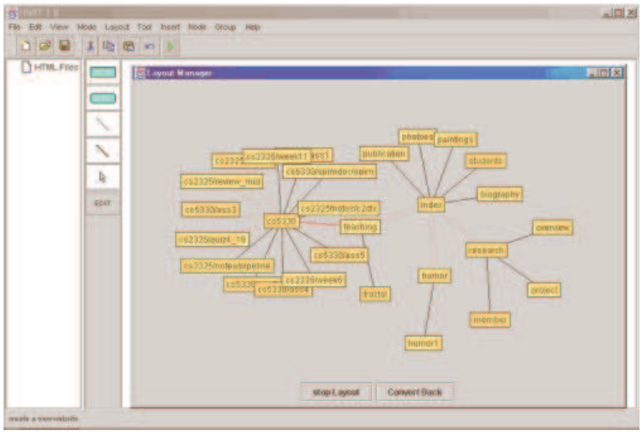


Figure 7: The layout process during reverse engineering of an existing Web site

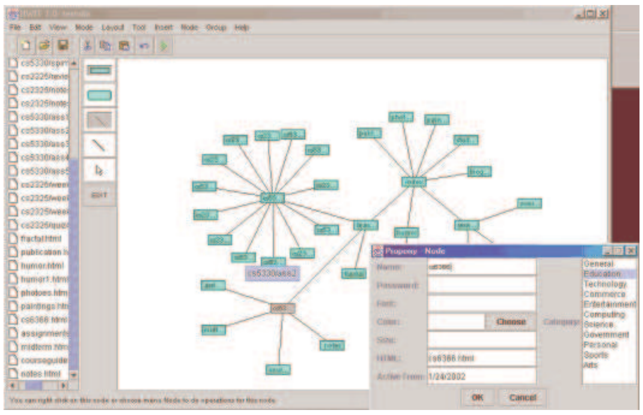


Figure 8: Enhancing the existing Web site by adding a sub-site

6. Conclusion and Future Work

This paper has presented a visual framework to Web site design, navigation, and maintenance. It advocates the integration of the tools for all activities, ranging from Web page and Web site design, navigation and browsing, to Web system maintenance, while preserving the same mental map for both the Web designer and the Web user throughout these activities. Our approach is a step closer towards narrowing the gap between Web designers and users [20].

We have implemented a prototype of HWIT in Java, which is capable of generating Web sites from Web graphs drawn on the HWIT Graph Editor. The Web re-engineering capability in HWIT allows previously developed Web sites to be visualized and re-developed graphically. Most of the presented features have been implemented. We plan to adapt an existing layout algorithm to support more personalized and pleasant viewing during navigation and maintenance.

This work has opened up many more opportunities for further investigation. Our future work will include the following.

- Security features will be built into the framework so that different groups of people may access different parts of a Web site.
- We will also conduct empirical studies in order to evaluate the usability of HWIT in real world applications.

References

[1] V. Anupam, J. Freire, B. Kumar, and D. Lieuwen, Automating Web Navigation with the WebVCR, *Proc. 9th Int'l World Wide Web Conf.*, Amsterdam, Netherlands, 15-19 May, 2000.

[2] G. Di Battista, P. Eades, R. Tammassia abd I.G. Tollis, Graph Drawing – Algorithms for the Visualization of Graphs, Prentice-Hall, 1999.

[3] B. Bederson, J. Hollan, J. Steward, D. Vick, L. Ring, E. Grose, and C. Forsythe, A Zooming Web Browser, *Human Factors in Web Development*, Eds. Ratner, Grose, and Forsythe, Lawrence Erlbaum Assoc. 1998, 255-266.

[4] S. Ceri, P. Fraternali, and A. Bongio, Web Modeling Language (WebML): A Modeling Language for Designing Web Sites, *Proc. 9th Int'l World Wide Web Conf.*, Amsterdam, Netherlands, 15-19 May, 2000.

[5] F. Coda *et al.*, Towards a Software Engineering Approach to Web Site Development, *Proc. 9th Int'l Workshop on Software Specification and Design*, IEEE Press, 1998.

[6] S. A. Conger and R. O. Mason, Planning and Designing Effective Web Sites, *Course Technology*, Cambridge, MA, 1998.

[7] P. Doemel, WebMap – A Graphical Hypertext Navigation Tool, *Proc. 2nd Int'l Conf. on the WWW*, USA, 1994, 785-789.

[8] P. Eades, A Heuristic for Graph Drawing, *Congressus Numerantium*,, Vol.42, 1984, 149-160.

[9] E. Frecon and G. Smith, WebPATH – A Three Dimensional Web History, *IEEE Symp. Information Visualization*, N. Carolina, October, 1998, http://davinci.infomatik.uni-kl.de/vis98/archive/fp/papers/webpath.html.

[10] J. Freire, B. Kumar, and D. Lieuwen, WebViews: Accessing Personalized Web Content and Services, *Proc. 10th Int'l World Wide Web Conf.*, Hong Kong, China, 1-5 May, 2001.

[11] G. Furnas, Generalized Fisheye Views, *Proceedings of CHI'86*, Boston, April 1986.

[12] J. Gómez, C. Cachero, and O. Pastor, Conceptual Modeling of Device-Independent Web Applications, *IEEE Multimedia*, April-June 2001, 26-39.

[13] S. Hinton, From Home Page to Home Site: Effective Web Resource Discovery at the ANU, *Proc. 7th Int'l World Wide Web Conf.*, 14-18 April 1998, Brisbane, Australia.

[14] M. L. Huang, P. Eades, and R.F. Cohen, WebOFDAV - Navigating and Visualizing the Web On-line with Animated Context Swapping, *Proc. 7th Int'l World Wide Web Conf.*, Brisbane, 14-18 April 1998.

[15] M. L. Huang, P. Eades, and J. Wang, On-line Animated Visualization of Huge Graphs Using a Modified Spring Algorithm, *J. Visual Languages and Computing*, 9, 1998, 623-645.

[16] R. Klapsing, G. Neumann, and W. Conen, Semantics in Web Engineering: Applying the Resource Description Framework, *IEEE Multimedia*, April-June 2001, 62-68.

[17] G.E. Krasner and S.T. Pope, A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, *JOOP*, 1(3), August 1988.

[18] Y.S. Maarek and I.Z.B. Shaul, WebCutter: A System for Dynamic and Tailorable Site Mapping, *Proc. 6th Int'l World Wide Web Conf.*, 1997, 713-722.

[19] D. C. Muchaluat, R. F. Rodrigues, and L. F. G. Soares, WWW Fisheye-View Graphical Browser, *Proc. IEEE of the 1998 Multimedia Modeling*, 1998.

[20] T. Nakayama, H. Kato, and Y. Yamano, Discovering the Gap Between Web Site Designers' Expectations and Users' Behavior, *Proc. 9th Int'l World Wide Web Conf.*, Amsterdam, Netherlands, 15-19 May, 2000.

[21] M. Sarkar and M. H. Brown, Graphical Fisheye Views, *Communications of the ACM*, Vol. 37 No. 12, December 1994..

[22] D. Siegel, *Creating Killer Web Sites: The Art of Third Generation Site Design, 2nd ed.*, October 1997, Hayden Books.

[23] M. B. Spring, E. Morse, and M. Heo, Multi Level Navigation of a Document Space, http://www.iis.pitt.edu/~spring/mlnds/mlnds/mlnds.html.

[24] W.W. Wadge and T. Yildirim, Intensional HTML, *Proc. 10th Int'l Symp. on Languages for Intensional Programming*, Victoria, Canada, 1997, 34-40.

[25] W3C, Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/REC-xml.html, Oct. 2000.

[26] D-Q. Zhang and K. Zhang, and J. Cao, A Context-Sensitive Graph Grammar Formalism for the Specification of Visual Languages, *The Computer Journal*, Vol.44, No.3, 2001, 186-200.

[27] K. Zhang, D-Q. Zhang, and J. Cao, Design, Construction, and Application of a Generic Visual Language Generation Environment, *IEEE Transactions on Software Engineering*, Vol.27, No.4, April 2001, 289-307.

[28] K. Zhang, D-Q. Zhang, and Y. Deng, A Visual Approach to XML Document Design and Transformation, *Proc. 2001 IEEE Symp. on Human-Centric Computing Languages and Environments*, Stresa, Italy, 5-7 September 2001, IEEE CS Press, 312-319.

[29] Y. Zhang and K. Zhang, Associative Query for Multi-version Web Documents, In: M. Gergatsoulis and P. Rondogiannis (Eds.) *Intensional Programming II*, World Scientific, 2000, 55-64.