Blitz's Mapster

Final SRS/SDS/Project Pan
(Group Three)

Team Members:

Jason Prideaux
Willy Suhali
Christian Tan
Hirokazu Yoshimura

TABLE OF CONTENTS

FIGURES

TABLES

Software Requirements Specification

Problem Statement

A web site with an enormous amount of content makes it difficult for a web author or web manager to visualize the content and hyperlinks for his or her site. A web page usually contains hyperlinks to other documents — both on the author's web site as well as to those on other web sites. Web authors can find it overwhelming to visually manage the contents of a site. This is especially apparent when the web site has a large amount of content. Furthermore, it is also an arduous task to manually check if the hyperlinks on all of the site's web pages are active or broken. Large web sites are often separated into different subdivisions, for which different teams of developers design each subdivision. A simple example would be that of the Gladstone server at the University of Oregon, where each user can have their own web page to design. Such segregation renders managing a web site an even more complicated undertaking.

Proposed Solution

Blitz's Mapster is a Java application is targeted towards web managers and web authors. It allows them to quickly access the tree layout of a web site based on an entry point of the Uniform Resource Locator (URL) and its subsequent hyperlinks. It also checks if the hyperlinks on the site are active or broken, and provides a list of broken hyperlinks. The main features of *Mapster* are:

- Runs virtually on any platform since it is a Java application.
- Provides a visual layout of the contents of a web site based on the entry point URL;
- Has an input field for user to specify how deep the search would go (for quick visualization of the layout);
- Checks if the hyperlinks on the pages of the site are active or broken;
- Provides a list of broken hyperlinks for easy management;
- Generates a quick summary of the site accessed, such as the number of files encountered;
- Distinguishes html files from other web-accessible documents by way of icons;
- Displays all hyperlinks from html files to other documents;
- Distinguishes relative links from absolute links; and
- Distinguishes external links (those to documents outside the site being mapped) from local links.

In the previous SRS, *Blitz* planned to include multiple views of the structure of the web site searched. However, due to time constraints, this trivial feature has been omitted to meet the release deadline of the application.

Scenario

Broken Hyperlinks

Joe is a web author and has a web site. It has been a while since he updated his web pages, and wonders if the hyperlinks on his web pages are still active. It would be a strenuous task for him to manually check the hyperlinks on each page on his web site. Instead, by putting the URL to his web site into *Blitz's Mapster*, and a click of the *Start Map* button, Joe is able to quickly access the status of the hyperlinks on his web site. By looking at the broken hyperlinks section of Mapster, Joe was able to quickly see which hyperlinks were broken, and was able to quickly fix those broken hyperlinks.

Visual Layout of Hyperlinks

Jane just joined *Macrohard, Inc.* as a web manager. She is assigned to manage the 'New Products' subdivision of the company's web site. Jane is also told that there are some files on the server that are no longer hyperlinked. Her first tasks are to remove all files that are no longer hyperlinked. Unfortunately, there is no documentation on the layout of the files and hyperlinks for the 'New Products' subdivision. Without documentation, Jane finds it difficult to visualize the layout of the site and the hyperlinks on the pages on this site.

With *Blitz's Mapster*, Jane is able to enter the URL of the site, and have the application map the site for her. After mapping, Jane was able to look at the tree structure produced to determine the files on the site. Comparing all the web-accessible files shown on *Mapster* and the files on the server, Jane is able to remove all the unwanted files with ease.

<u>Specification</u>

In the previous SRS, *Blitz* planned to have a summary button that would pop up a new window showing the summary of the site mapped. However, after much thought, Blitz decided that it would be more useful to the user if the user can dynamically see the summary of the site as it is being mapped. Hence, a separate Summary tab has been added to the System. Also, in place of the summary button, *Blitz* has added a *Stop Mapping* button that allows the user to stop a current search. The improved *Mapster* has the following features (see Figure 1):

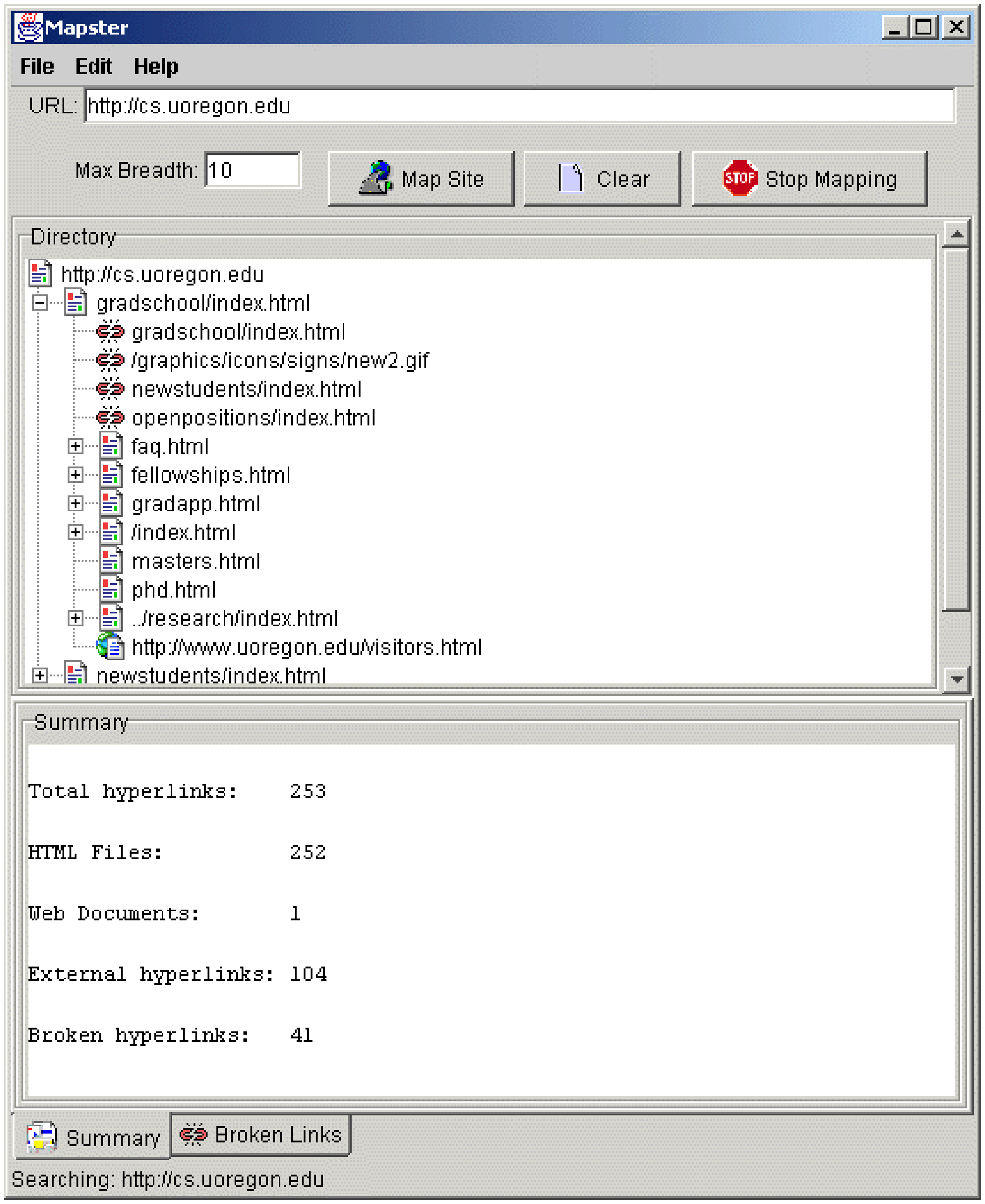
- Pull-down menu
- *Entry Point URL* textbox field
- *Maximum Breadth* textbox field
- *Map Site* button
- *Clear* button
- *Stop Mapping* button
- Layout window
- *Broken Links* tab window
- *Summary* tab window
- *Icons*

Pull-down menu

Mapster has the familiar pull-down menu for actions such as Copy, Paste, and Quit Application.

Entry Point URL textbox field

This textbox field is where the user enters the entry point URL from which *Mapster* is to start its search. When the user presses the *Map Site* button, Mapster will check if the field is empty, or if it is accessible, and alerts the user of any errors prior to mapping. It also checks for a protocol. If no protocol is entered, the default *"http://"* will be appended to it.



```
Total hyperlinks:    253

HTML Files:          252

Web Documents:       1

External hyperlinks: 104

Broken hyperlinks:   41
```

Figure 1. *Blitz's Mapster* Application

Maximum Breadth **textbox field**

This textbox field is where the user enters the maximum breadth for which *Mapster* is to search. Since web sites can get fairly large, a limit on the breadth of the search can lessen the wait time for quick checks on a web site. The user is alerted if an invalid number has been entered.

Map Site **button**

When pressed, *Mapster* begins its search using the value entered in the *Entry Point URL* textbox field. This button is disabled once it is pressed to prevent the user from pressing it twice during a search.

Clear button

When pressed, the *Entry Point URL* textbox field, layout window, and the *Broken Hyperlinks* window are cleared for the next search. The default maximum breadth is also restored. This button is disabled once the *Map Site* button is pressed to prevent the user from pressing it during a search.

Stop Mapping button

When pressed, *Mapster* would terminate mapping the site. This allows the user to cancel any search to start another search. This button is only enabled when the *Map Site* button is pressed and is disabled once the search is complete or when the user presses the Stop Mapping button.

Layout window

This is the window where the layout of the site is displayed. Since the builder of the site is threaded, this layout window is dynamically updated when the hyperlinks on each page is obtained and processed. This feature is an improvement based on testing of the system, as previous builds did not have any display until the site was completed, which can take up to 10 minutes. The layout window also uses icons to categorize the hyperlinks obtained (see below), as well as tool-tip text to let the user quickly understand what each icon means.

Icons

The Layout window uses a tree structure to display the relationship of web documents. *Mapster* utilizes icons to categorize the type of documents obtained from the hyperlinks found. The following table shows the type of icons and their description

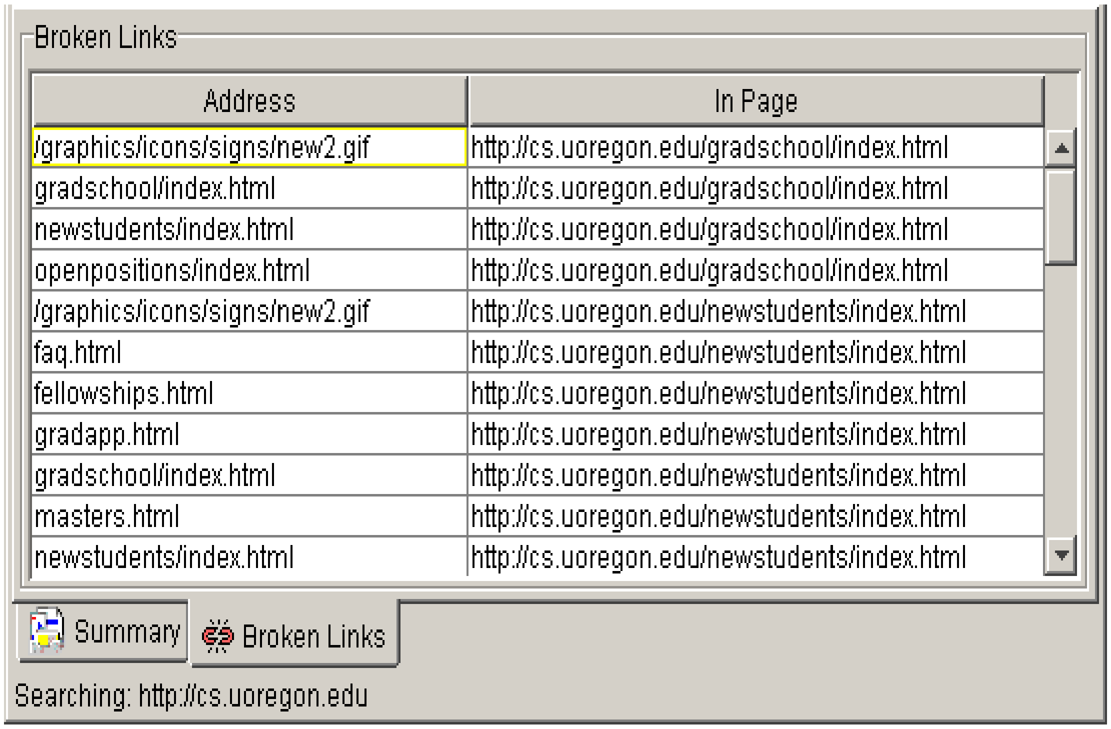
Icon	**Description**
🔗	This icon represents a broken hyperlink. This can represent any type of document (HTML, external, image, audio, etc.) that is not accessible by *Mapster*.
🌐	This icon represents an external hyperlink. External hyperlinks are not mapped.
🔒	This icon represents a secured document, going though Secured Socket Layer (SSL). Such documents are not mapped for security reasons.
📄	This icon represents HTML files on the local server. This icon is expandable (depending on the maximum breadth limit) to contain other documents.
🖼	This icon represents image files.
📁	This icon represents all other web-accessible documents on the local server.

Table 1. Description Of The Layout Window Icons

***Broken Links* tab window**

In the previous design, Blitz had proposed a *Broken Links* window to have four columns: Status, Hyperlink, In Page, and Page Title. However, upon revision, it is decided that the *Status* and *Page Title* column were trivial and redundant. Hence the new broken Links tab window has only the following two columns (See Figure 2):

1. The *Address* column displays the broken hyperlinks.
2. The *In Page* column displays the location of the web page where the hyperlink is broken.



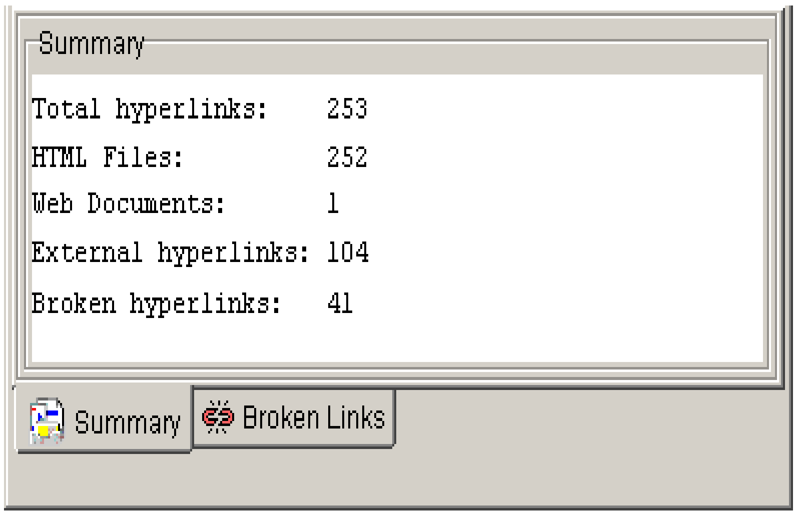
Address	In Page
/graphics/icons/signs/new2.gif	http://cs.uoregon.edu/gradschool/index.html
gradschool/index.html	http://cs.uoregon.edu/gradschool/index.html
newstudents/index.html	http://cs.uoregon.edu/gradschool/index.html
openpositions/index.html	http://cs.uoregon.edu/gradschool/index.html
/graphics/icons/signs/new2.gif	http://cs.uoregon.edu/newstudents/index.html
faq.html	http://cs.uoregon.edu/newstudents/index.html
fellowships.html	http://cs.uoregon.edu/newstudents/index.html
gradapp.html	http://cs.uoregon.edu/newstudents/index.html
gradschool/index.html	http://cs.uoregon.edu/newstudents/index.html
masters.html	http://cs.uoregon.edu/newstudents/index.html
newstudents/index.html	http://cs.uoregon.edu/newstudents/index.html

Searching: http://cs.uoregon.edu

Figure 2. Broken Links Tab Window

***Summary* tab window**

This window dynamically displays the summary of the site as it is being mapped. This tab window is a feature improvement from the previously proposed pop-up summary window. This window displays information such as the total number of hyperlinks encountered, the number of those hyperlinks being HTML files, the number of web-accessible documents (such as images, audio files, downloadable files, etc.), the number of external hyperlinks, and the number of broken hyperlinks (See **Error! Reference source not found.**).



Summary

```
Total hyperlinks:    253
HTML Files:          252
Web Documents:       1
External hyperlinks: 104
Broken hyperlinks:   41
```

Figure 3. Summary Tab Window

Additional Requirements:

Mapster is a Java application, and hence is platform independent. Since it is a Java application, it requires the user to have Java version 1.3 to be already installed on the computer. Also, since *Mapster* queries web sites, an Internet connection must already be established.

Known Bugs:

In previous SRS and SDS, *Blitz* planned to use a freeware API to aid in the development of *Mapster*. However, after much testing, we discovered that the freeware API was incompatible with our system, and it delayed our project significantly. For that, we had to build the spider and HTML Parser from scratch, and due to limited time constraints, some of the following bugs have yet to be ironed out:

- Some hyperlinks are displayed as broken, but are not. This problem is inherit in the URLParser class. This problem arise when the URLConnection class (Java API) returns a header that is not of the form "HTTP/1.0 200 OK" which is a protocol that signals that the web document is located on the server.

- Hyperlinks that were visited are searched again redundantly. This problem is inherit in the DataBuilder class.

- When searching particular websites, some hyperlinks are not displayed even though they exist on the page. This problem is inherit in the DocFetcher class. This problem arises in using the URLConnection class, as it sometimes returns only part of the entire document.

- Javascripts and dynamic pages are not supported. This is a design choice made by *Blitz*. Future upgrades of this system may include dynamic pages and javascript links to other pages.

Software Design Specifications

Design: high level view

In the Software Requirements Specifications (SRS) it is clear that the required system needs to take in a Uniform Resource Locator (URL) and create a data structure, which is to be displayed on a user interface window. Given this basic description of the product, the design consists of modules that are built to perform specific tasks that all work together to create the functionality of *Blitz's Mapster*.

The different tasks required of *Mapster* are fetching documents from a URL, creating a data structure from the aforementioned documents, and a user interface to display the data structure to the intended user.

The abstract structure of *Mapster* consists of four main modules, which will make up the system. These four main modules are Fetcher, Builder, Data Structure, and Graphical User Interface (GUI). Some of these modules will contain multiple components, such as the Data Structure, which contains the components, WebTree and Node. The system components are depicted in Figure 2, where the arrows represent the flow of data. The GUI sends a URL to the builder, who sends the URL to the Fetcher, who returns a list of hyperlinks. Then the Builder calls on UrlParser to get information regarding the URL. Next Builder creates Nodes, which represent the hyperlinks, and these Nodes are sent to the WebTree. Finally the Builder contains a tree structure in the form of a WebTree; Builder then takes the tree structure and sends it to the GUI, who adds the tree structure to a displayable JTree.

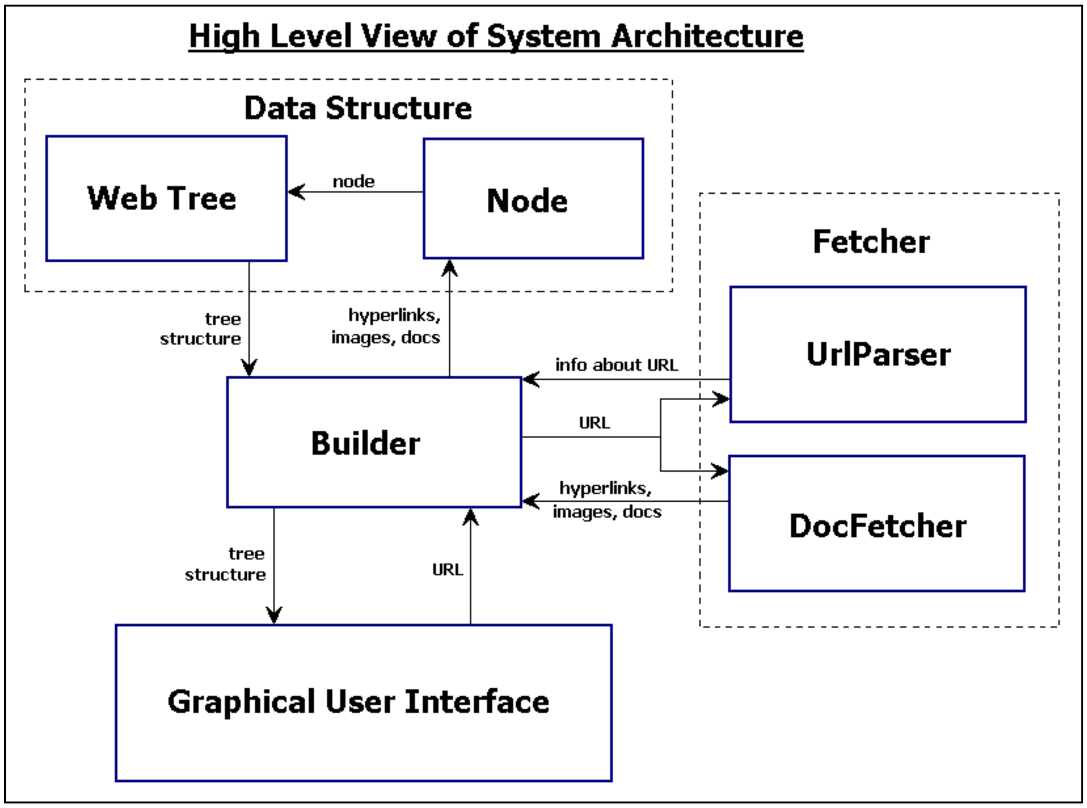


Figure 4. Interaction of data between system modules

Structure: View of Individual Components

Blitz's Mapster is composed of four main modules: Fetcher, Builder, Data Structure, and GUI. In addition, Data Structure is composed of five components: BrokenLink, Node, NodeComparator, NodeRenderer, and WebTree. Fetcher is composed of DocFetcher and UrlParser. Figure 4 on the previous page gives a object model of these different components and shows in more detail the inner attributes and functions of each component. In addition, Figure 3 on the following page is a low-level view of each component. Each box displays the component name, the information stored in the component, and the component's primary functions/methods.

Fetcher:

Doc Fetcher

The DocFetcher is the component that takes a URL, and returns a list of hyperlinks in the URL. The DocFetcher grabs the html content from a URL, and then proceeds by parsing the html and finding all hyperlinks on that page.

UrlParser

This component is primarily a helper class. It is a static class that has the purpose of providing a set of helper methods for the Builder module. When the Builder is creating Nodes out of hyperlinks returned from the DocFetcher, it needs to gather information such as whether the link is broken, external, absolute, or relative. The UrlParser will take in the URL of the link and return to the builder pertinent information regarding the URL.

Builder:

DataBuilder

The DataBuilder takes URLs from the GUI. It then uses this information to grab the contents of the URL by calling upon DocFetcher. The DataBuilder then takes the hyperlinks returned by the DocFetcher, and builds Nodes for each hyperlink. The Node object will contain all necessary information about the hyperlink obtained from the UrlParser. Next the DataBuilder puts the Nodes into a WebTree object and will then repeat the process by calling DocFetcher on the URL of the new Nodes until a limit set by the user is reached. In addition, DataBuilder is now threaded so that the user won't have to wait so long to view the tree. Once the first level of the WebTree is complete, the user may view it, then when additional sections of the tree are complete, the notifies the GUI, and a 'plus' symbol will appear by a node which now contains more information.

Data Structure:

Node

This component is the representation of an individual document or html of a web site. Nodes which represent a html will, of course, contain other documents or links, and so those Nodes with links to other Nodes that represent the documents and HTMLs inside that Node. On a side note, Node is accompanied by three additional classes: NodeComparator, NodeRenderer, and BrokenLink. These classes are merely assistants to the main Node class. They provide features such as comparing nodes, set the icon of a node, and classifying if a node represents a broken link or not.

WebTree

This component represents the entire structure of a URL. It will take in Node objects and serve as a means for the GUI to understand the Node structure.

NodeComparator

This component is a comparator created to compare two Nodes if one is less than, equal to, or greater than another Node. This is used mainly in sorting the hyperlinks obtained.

NodeRenderer

This component used by the JTree to render the icon images in the layout window of the GUI. It also sets the Tool-Tip text based on the Node information.

BrokenLink

This class holds information about each broken link encountered. This is created in the DataBuilder, and is used by the *Broken Links* tab window on the GUI.

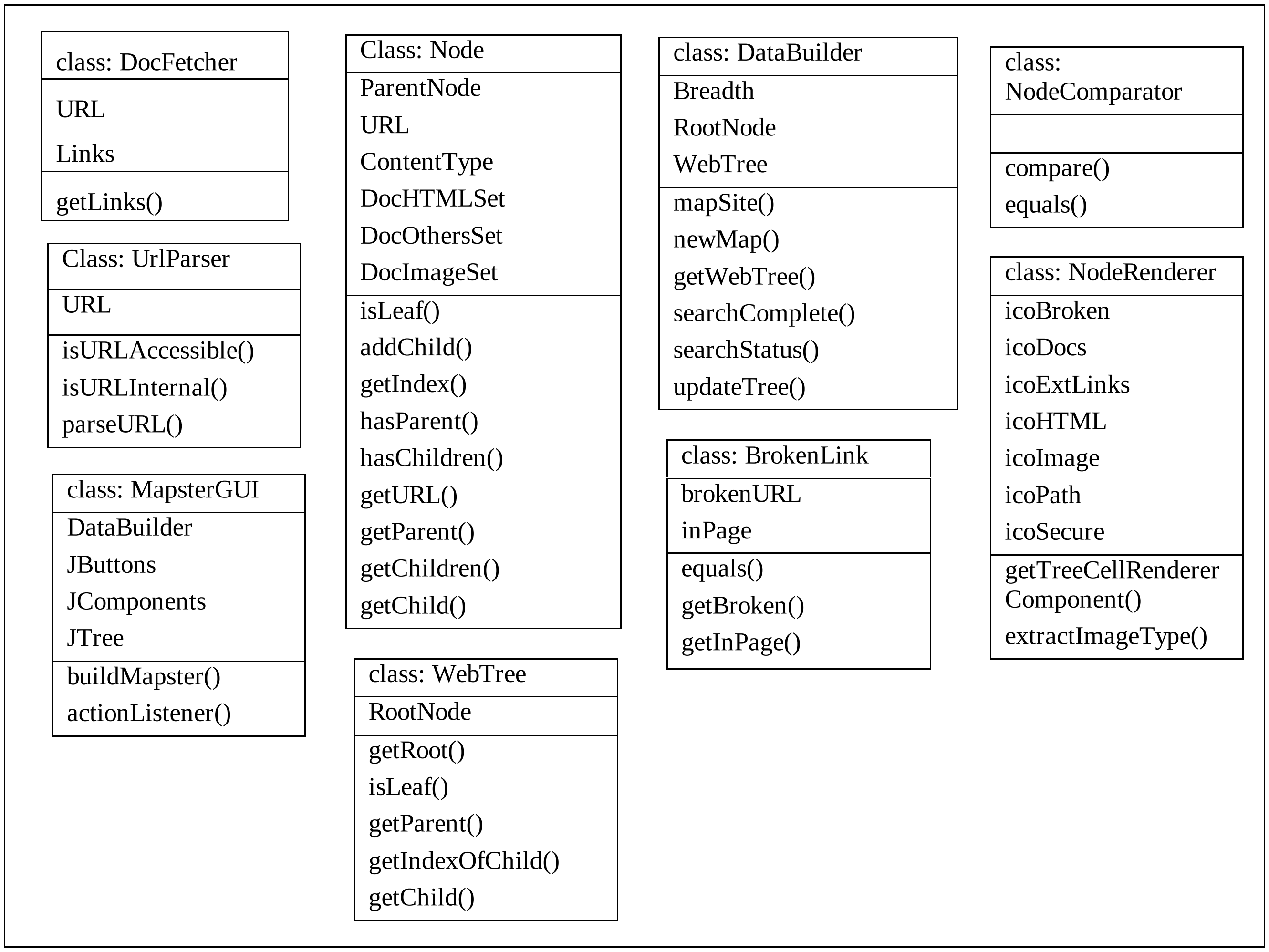


Figure 5. Object Model of System components

Graphical User Interface (GUI):

MapsterGUI

This is the most important part of the *Mapster* software program, since it is what interacts with the user. The GUI takes information from the user, such as the URL and the breadth size of the web site mapping. In addition, the user can dynamically view sections of the web site as shown in Figure 1 of the SRS (Page 3). There is also a separate window to display broken hyperlinks and the summary of links, which the Builder module keeps track of. In addition, user buttons, such as *Clear* will reset all data in the Builder, and *Map Site* will begin a new web site mapping by sending a new URL to the DataBuilder.

Changes to the Original Design

Originally it was believed that freeware API could be used in the construction of *Mapster*, particularly for retrieving and parsing html documents. However, after thorough consideration and vigorous testing, all members of *Blitz* agreed that the freeware APIs and components under consideration could not perform any of the necessary tasks to our satisfaction. Thus, modules such as Fetcher, Builder, and Data Structure went through a redesign phase in order to make accommodations for the change in html parsing.

Organization Principles

The overall organization of the system modules is based on a structural pattern. In this design pattern the primary concern with the system is how the modules will be composed to create the main system. Since there are four primary modules, the structure is composed of many components grouped inside modules, this allows related components to communicate easily, and provides an easy way to modify one component without affecting the other components.

As for the organization of the building process, *Blitz* uses the Spiral Process model as shown in the Project Plan (page 11). While time is a major constraint, it was a necessity to have a process, which allows for revisiting the software requirements and design. During some phases of redesigning, it was not necessary to change the entire system; only problem modules need modification. Thus, the system design allowed for flexibility during the development.

Rationale

The design of this system benefits both the client as well as developers. The system, being comprised of four main separate modules, enables *Blitz* to develop the modules in parallel. Thus, the modules were built by separate programmers, which made delivering the system on time possible. This resulted in the client receiving the system in a short amount of time, as opposed to the system being built one piece at a time.

The issue of reusing freeware code for this project is an example in favor of the system design. Originally, it was believed that reusing freeware code would take the place of the Fetcher module of *Mapster*. However, when it turned out that the freeware code was not performing to expectations, a total rewrite of the system was not necessary. In fact, only the Fetcher module had to be reworked. Thus, with the use of modules, it is relatively easy to replace, test, and modify individual components without affecting other components.

In summary, due to the design, testing was performed on the individual components before the final system integration, to help ensure the quality of the product. In addition, if developers want to add additional functionality to the system, it can be done in a clean and easy manner, such as explained in the previous paragraph. Additional modules can be added to system, or a particular component can be modified without having to change the entire system.

Project Plan

Project Organization

Organizational Structure

Blitz was coordinated and directed by the project manager, Jason Prideaux, who is also responsible for administrating the project schedule. *Blitz* made decisions unanimously — all decisions were considered thoroughly and clearly, and agreed on by every *Blitz* team members.

Team Responsibilities

Blitz's responsibilities were divided into seven team roles: project manager, quality control, system architect, technical documentation, user documentation, user interface, and configuration control. Each role is assigned in favor of the members' skills and experiences. There were at least 2 members assigned to each role. This ensured that when the primary member faced any difficulties in the role, the secondary member was be able to help solve the problem. Table 2 presents the members' roles and responsibilities for each task.

In addition, for the roles of programming, the team divided into two groups. The GUI was one large programming task and was tackled by Kaz and Willy. Jason and Christian programmed the other three modules, being so tightly connected. Having teams of two working on system components aided in the development process greatly. The ideas of the two-person teams helped to avoid brain-freezes, and resulted in a faster generation of ideas. This led to an extreme-programming approach where much of the functionality of modules was determined during the development process of the software.

Roles	**Primary**	**Secondary**
Project Manager	Jason Prideaux	Christian Tan
Quality Control	Hirokazu Yoshimura	Jason Prideaux
System Architect	Jason Prideaux	Willy Suhali
Technical Documentation	Christian Tan	Hirokazu Yoshimura
User Documentation	Willy Suhali	Christian Tan
User Interface	Hirokazu Yoshimura	Willy Suhali
Configuration control	Christian Tan	Jason Prideaux

Table 2. Team Role Assignments

Team Communication

Blitz had meetings at least 3-5 times a week, and towards the last two weeks before completion, the meetings were held daily. The meetings were held at the EMU International Resource Center and at the Deschutes computer lab. The progress report was checked against the project plan and milestones, and was updated accordingly during the meetings or via E-mail by the primary and secondary Project Managers.

Risk Analysis and Risk Reduction Strategies

Risks are prevalent in software development and may impede *Blitz* from delivering the software on time. To solve this matter, *Blitz* used risk analysis and risk reduction strategies. Table 3 presents some possible risks that might and did occur during the development, and the corresponding plans to handle those risks.

Risk	Probability	Effects	Strategy
Falling behind the schedule	High (happened)	Serious	Delivering regularly and as soon as possible; milestone are strictly enforced. The Project Manager keep the team as focused as possible.
Loss of a team member.	Moderate	Serious	Assigning two members for each task, the secondary member will still be working on the task if the primary person got sick or leaves the team.
Diverging away from the requirements	High (happened)	Serious	Try to come out with prototype early, use the spiral model software process. The design of the system underwent many changes. Working overtime was necessary to overcome this hurdle.
Conflict between team members	High (happened)	Serious	Manage conflicts constructively; try to make a unanimous decision or consult a higher authority, such as the Professor. After thoughtful consideration by all members, all conflicts were managed.

Table 3. Risk Analysis and Risk Reduction

Work Breakdown and Project Schedule

Milestones

The project had thirteen main milestones as described in Table 4 below, some of which contain multiple tasks to be completed. The Progress Report on the last page shows the dates of actual completion. When the actual coding began, several problems were encountered (such as the freeware API discussed in the SDS) and as a result the milestones after Jan 24 were subsequently delayed.

Milestones	Dates
Start the project plan	Jan 14, 2002
Finalize SRS/SDS/Project Plan	Jan 16, 2002
Preparing presentation material	Jan 20, 2002
Begin construction on GUI module Begin construction on WebTree module Begin construction on Node module Begin construction on Tree Builder module	Jan 20, 2002
Complete GUI module Complete WebTree module Complete Node module Complete Tree Builder module	Jan 24, 2002

Testing of all 4 modules separately	Jan 25, 2002
Integrating the 4 modules into one system	Jan 26, 2002
Testing of the entire system	Jan 27, 2002
Finalizing and getting a deliverable system	Jan 29, 2002
Finishing user documentation and technical documentation	Jan 30, 2002
Finalizing the SDS/SRS/Project plan	Jan 31, 2002
Product delivered	Feb 4, 2002
Final presentation	Feb 5, 2002

Table 4. Project Milestones

Project Process Breakdown

The software process for this project followed a Spiral Process (See Figure 6). There was six main phases in the development of the software, however, given the high possibility in change of requirements or design, it is quite possible that some of these phases will be revisited. In fact, many of these phases were revisited and explained in the following descriptions of each process phase.

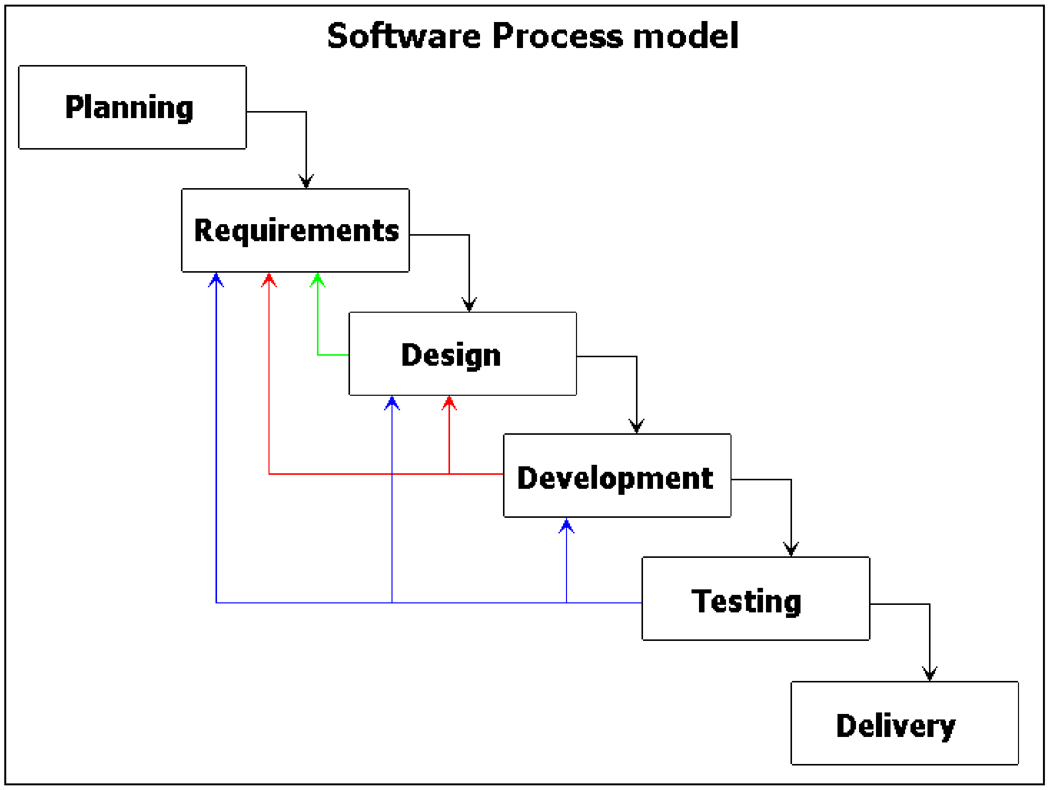


Figure 6. The Spiral Process Model

Planning

The planning phase included: setting up the project schedule, delegating responsibilities/roles, and defining the problem statement for this project.

Requirement analysis

The requirement analysis phase included analyzing the problem statement to create our list of requirements as described in the SRS. The list of requirements was then used to describe the design of the system. This phase of our process was visited more than once, but only in order to make the requirements of the system more specific, and to remove any ambiguity.

Design

The design phase included: creating a system architecture, identifying our 4 main modules and sub-components, and specifying when modules should be built and by whom. This phase was revisited quite often throughout the project timeline. The design of the User Interface went through many internal changes, relating to how to program the desired features. Also as discussed in the SDS page 10, the reuse of code that was in the original design, was omitted. This resulted in a major redesign of the Data Structure, and Data Builder modules.

Development

At this phase, every group member had a very specific programming task, and through regular meetings, the progress of each group member was monitored. At a scheduled date, all code was integrated to create the system. In addition, during this phase is when the requirement and design phases where revisited. Problems, such as the reuse of code would be discovered in this phase, and thus requirements or design features would be updated accordingly.

Testing

Testing was to be done by individual group members at the completion of his or her module. After integration of the system two group members performed module testing.

Delivery

Once the system was completed, all group member were given a short amount of time to test, review the system, as well as add or change parts of it.

Monitor and Report Progress

Blitz kept a record of each task assigned, to whom the task was assigned, when they it was assigned, the scheduled due date of the task, and when the task was completed. *Blitz's* members signed off on the progress report when each task was completed and verified by the other members.

The following table is the list of tasks that were required to create software system. This is similar to the intended schedule of milestones in Table 4., but it differs in that some dates of tasks were not met, which delayed many of the succeeding milestones.

Task	**Assigned to and completed by**	**Date assigned**	**Scheduled completion**	**Actual Completion**
Meeting: Start SRS/SDS/Project plan	ALL	Jan 14	Jan 14	Jan 14

Finalize SRS	Christian	Jan 14	Jan 17	Jan 20
Finalize SDS	Jason	Jan 14	Jan 17	Jan 20
Finalize Project Plan	Willy	Jan 14	Jan 17	Jan 20
Meeting: gather SRS/SDS/Project plan	ALL	Jan 17	Jan 17	Jan 17
Finalize Presentation materials	Kaz	Jan 17	Jan 21	Jan 22
Meeting: Prepare for presentation	ALL	Jan 22	Jan 22	Jan 22
Design system architecture	Jason	Jan 17	Jan 22	Jan 22
Meeting: delegate programming tasks	ALL	Jan 22	Jan 22	Jan 22
Begin the GUI	Kaz/Willy	Jan 22	Jan 25	Jan 25
Begin DataBuilder/Fetcher	Jason	Jan 22	Jan 25	Jan 25
Begin Data Structure components (Node, WebTree)	Christian	Jan 22	Jan 25	Jan 25
Meeting: review progress	ALL	Jan 25	Jan 25	Jan 25
Meeting: review progress/prototypes	ALL	Jan 28	Jan 28	Jan 28
Compete GUI	Kaz/Willy	Jan 28	Jan 31	Feb
Begin User Documentation	Kaz/Willy	Jan 28	Feb 1	Feb
Compete Fetcher/Builder	Jason	Jan 28	Feb 1	Feb
Complete Node/WebTree	Christian	Jan 28	Jan 31	Jan 31
Integration of the 4 modules	Jason/Christian	Jan 30	Jan 30	Jan 31
Testing of the entire system	ALL	Jan 30	Feb 1	Feb 2
Finalizing the SDS/SRS/Project plan	Jason/Christian	Jan 30	Feb 4	Feb 4
Finishing user documentation	Kaz/Willy	Jan 30	Feb 2	Feb 4
Beginning & finishing technical documentation	Christian	Jan 31	Feb 3	Feb 4
Finalizing and getting a deliverable system	Jason/Christian	Feb 1	Feb 3	Feb 4
Product delivered	ALL	Feb 4	Feb 4	Feb 4
Final presentation	ALL	Feb 5	Feb 5	Feb 5

Table 5. Project Tasks