

Mihai Christodorescu
mihai@cs.ucsb.edu
Donald Adams
sbdon@cs.ucsb.edu
Michael Weaver
deckard@cs.ucsb.edu

SlothWare

Project #8

Wednesday, February 17, 1999

Our software ain't fast, but it sure is slow!

Computer Science 172
Software Engineering

Peter Capello
Matthias Koelsch
Mike Neary

WebShell Object–Oriented Analysis

version 1.0

Tuesday, February 16, 1999

Analyzing the functionality desired in the WebShell project, we came up with the following classes, each encapsulating independent amounts of information:

- ➔ the **GUI** class that hides all the details related to maintaining a tree and a list view
- ➔ the **Console** class that hides all the details related to parsing, interpreting and displaying commands and their output
- ➔ the **HTTPfs** (HTTP File System) class that will make the web pages look like directories on the local file system

The main program (which is in the Sloth class) will create the **GUI** and the **Console** windows. Applying the Model–View–Controller pattern, we came up with another class, the **Control** class, that will coordinate the actions and data exchanges between the **GUI** and the **Console**, on one hand, and the **HTTPfs**, on the other hand.

To make the **Control** class more general, we imposed a generic interface on the **GUI** and on the **Console**, the **View** interface, that unifies and provides the minimum of functionality and of type information.

Since the requirements changed to include browsing of the local file system, we similarly unified the **HTTPfs** and the **Localfs** under one interface, the **FileSystem** interface. This **FileSystem** interface provides functionality to query about the files and the file types existing in the current directory.

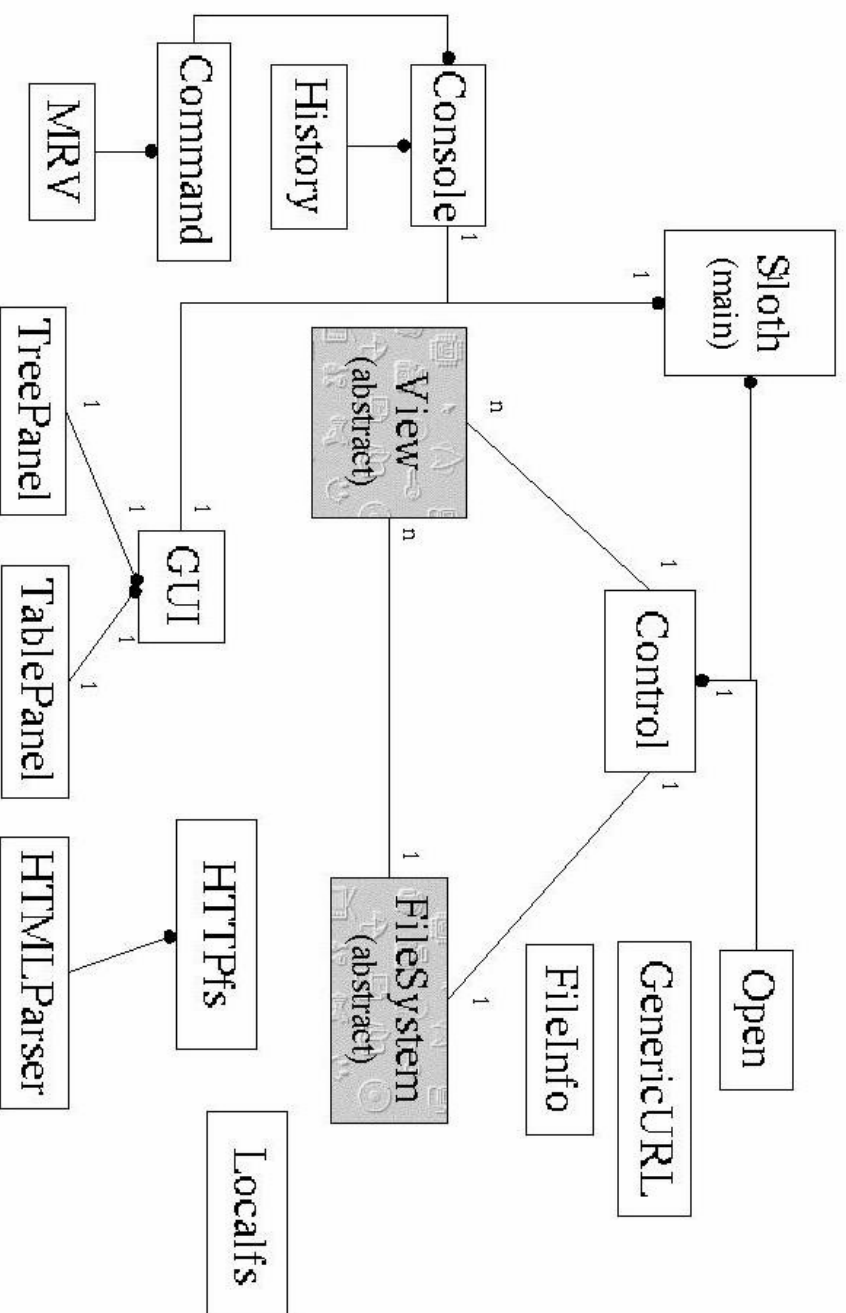
Because we were unsatisfied with the performance and the inconsistent functionality of the Java library class `java.net.URL`, we created our own URL handling class, called **GenericURL**, which is completely RFC–compliant. The **GenericURL** class is used by the **Control** and the **Views** to transfer and display file system information.

The **FileInfo** class acts as a data encapsulation unit, without any data hiding, its mission being that of allowing for the easy transfer and query of information from the **FileSystem** to the **Views** and the **Control**.

The **Open** class handles the open request of the WebShell, hiding the ugliness of system-dependent program invocation from the **Control**.

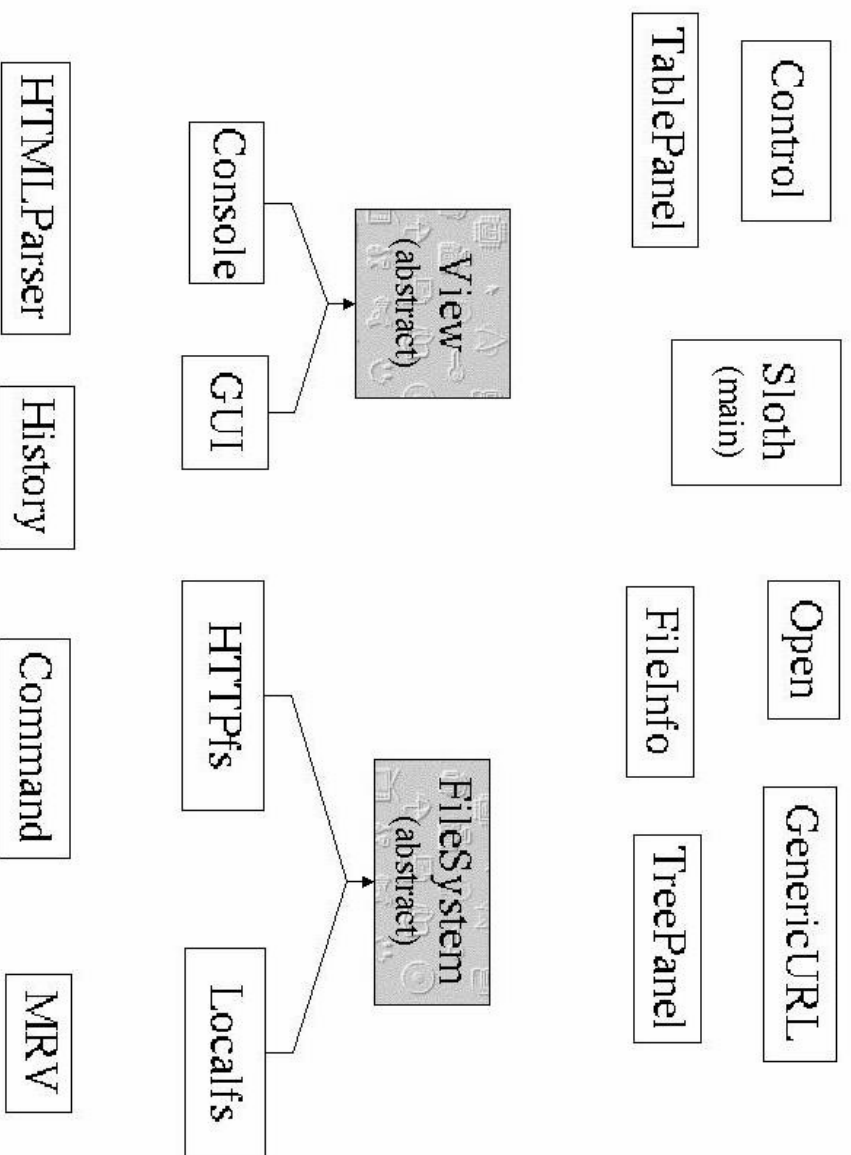
The **GUI** and the **Console**, as well as the **HTTPfs**, use helper classes to perform their required functionality. That is how the **TablePanel**, **TreePanel**, **Command**, **MRV**, and the **HTMLParser** were created.

Object Diagram for the WebShell Project



by SlothWare

Class Diagram for the WebShell Project



by *SlothWare*

Contents

Product Constraints.....	5
Black Box Specification.....	6
Input.....	6
Output.....	6
Input/Output Functional Relationships.....	6
Acceptance Criteria.....	6
Class Descriptions.....	7
Class Command.....	7
Black Box Specification.....	7
Input–Output.....	7
Use Cases.....	8
Scenarios.....	9
CRC.....	9
Class Console.....	10
Black Box Specification.....	10
Input–Output.....	10
Use Cases.....	11
Scenarios.....	12
CRC.....	13
Class History.....	13
Black Box Specification.....	13
Input–Output.....	13
Use Cases.....	14
Scenarios.....	14
CRC.....	14
Class MRV.....	15
Black Box Specification.....	15
Input–Output.....	15

¹ This is a preliminary version. Use of this documents is for informational and non–commercial or personal use only and this document will not be copied or posted on any network computer or broadcast in any media. SlothWare makes no representations about the suitability op the information contained in the document published herein for any purpose. This document is provided "as is" without warranty of any kind. SlothWare hereby disclaims all warranties and conditions with regard to this information, including all implied warranties and conditions of merchantability, fitness for a particular purpose, title and non–infringement. In no event shall SlothWare be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available in this document. The document herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein.

Use Cases.....	15
Scenarios.....	16
CRC.....	16
Class Sloth.....	16
Black Box Specification.....	16
Input–Output.....	16
Acceptance Criteria.....	16
Use Cases.....	17
Scenarios.....	17
CRC Card.....	18
Class TreePanel.....	18
Black Box Specification.....	18
Input–Output.....	18
Acceptance Criteria.....	18
Use Cases.....	19
Scenarios.....	19
CRC Cards.....	19
Class TablePanel.....	20
Black Box Specification.....	20
Input–Output.....	20
Acceptance Criteria.....	20
Use Cases.....	20
Scenarios.....	20
CRC Cards.....	21
Class GUI.....	21
Black Box Specification.....	21
Input–Output.....	21
Acceptance Criteria.....	21
Use Cases.....	22
Scenarios.....	22
CRC Cards.....	22
Class Control.....	23
Black Box Specification.....	23
Input.....	23
Output.....	23
I/O Relationship.....	23
Use Cases.....	24
Scenarios.....	24
CRC Card.....	24
Class Open.....	25
Black Box Specification.....	25
Input.....	25
Output.....	25
I/O Relationships.....	25
Use Cases.....	25
Scenarios.....	25
CRC Card.....	25
Class FileSystem.....	26

Black Box Specification.....	26
Input.....	26
Output.....	26
I/O Relationships.....	26
Use Cases.....	26
Scenarios.....	27
CRC Card.....	27
Class HTTPfs.....	27
Black Box Specification.....	27
Input and Output.....	27
Use Cases.....	28
Scenarios.....	28
CRC Card.....	28
Class Localfs.....	28
Black Box Specification.....	28
Input and Output.....	28
Use Cases.....	28
Scenarios.....	28
CRC Card.....	28
Class FileInfo.....	29
Black Box Specification.....	29
Input.....	29
Output.....	29
I/O Relationships.....	29
Use Cases.....	29
Scenarios.....	29
CRC Card.....	29
Class GenericURL.....	30
Black Box Specification.....	30
Input.....	30
Output.....	30
I/O Relationships.....	30
Use Cases.....	30
Scenarios.....	30
CRC Card.....	30
Class HTMLParser.....	31
Black Box Specification.....	31
Input.....	31
Output.....	31
I/O Relationships.....	31
Use Cases.....	31
Scenarios.....	31
CRC Card.....	32
Class View.....	32
Black Box Specification.....	32
Input.....	32
Output.....	32
I/O Relationships.....	32

Use Cases.....	32
Scenarios.....	32
CRC Card.....	33

Product Constraints

The deadline of the product is March 18th, 1999.

The product is planned to be as hardware- and software- independent as possible. Since it is written in Java, it will be portable across platforms, with the only foreseeable restriction being the fact that external applications are spawned in a system-dependent manner.

Black Box Specification

The WebShell will provide a new paradigm in web based computing. It allows access to a remote web page as if it were a local file system.

The WebShell will provide two ways to access the web pages, through a standard shell interface, and a GUI-based interface. The GUI interface allows the user to browse the remote web page, using a point-and-click metaphor. The WebShell enhances the user experience by allowing external application to be executed when the user needs to view a specific data type not handled by the shell itself. The program will also allow browsing the local file system.

Input

The inputs to be WebShell are provide through a dual hardware interface: a keyboard and a mouse. The WebShell takes commands from the keyboard (from the set cd, up, back, forward, ls, info, open, set, alias, and unalias) and executes them, updating the windows accordingly. Also, using the mouse, the user can browse the document tree.

Output

The output of the WebShell is two-fold: on one hand, the console window provides limited output (information and error text), on the other hand, the GUI displays a tree rooted at the current directory, and a list of files in the current directory.

Input/Output Functional Relationships

When a user enters a command that affects the display (cd, up, back, or forward) the windows will be updated to reflect the change in directory structure. When a user clicks on an entry in the tree, the current directory is changed in both the GUI and the console. Double clicking on an item in the GUI, or typing the open command with and argument, will open the pre-defined handler application for that URL/file type.

Acceptance Criteria

The product is considered feature–complete and ready to ship when all the functionality described above is implemented in full, and all the (major) bugs are eliminated. User testing of the product will offer additional feedback before considering the product completed.

Use Cases and Scenarios

The use cases and scenarios span from the specific use cases and scenarios from the top-level classes Control, FileSystem, GUI, and Console.

Class Descriptions

Class Command

Black Box Specification

The Command class provides the functionality of the console commands to the Web Shell console.

The Command class provides the functionality of the console commands to the Web Shell console. Certain commands are implemented entirely within this class while others rely on calls to external modules like the history list.

Input–Output

The following is a list of public methods in the Command class.

ChangeDirectory(String path)

The ChangeDirectory method takes a string as a parameter. It changes the current directory to string path and returns the new absolute path, if it is valid. If the path is invalid an error message is returned. If path is null then ChangeDirectory does nothing.

Up

The Up method changes the current directory to be the current directory's parent and returns the new absolute path. If the current directory does not have a parent then Up returns an error message.

Back

The Back method changes the current directory to be the previous directory as determined by the MRV–queue and returns the new absolute path. If there is no previous directory then Back returns an error message.

Forward

The Forward method changes the current directory to be the next directory as determined by the MRV–queue and returns the new absolute path. If there is no next directory then Forward returns an error message.

ListDirectory(String path)

The ListDirectory method returns a list of the contents of the directory path. If path is null then ListDirectory returns a list of the contents of the current directory. If path is invalid or there is no current directory then ListDirectory returns an error message.

Info(String document)

The Info method returns information about the document. If document is null or invalid Info returns an error message.

Open(String document)

The Open method spawns a helper app to view the document. If document is null or invalid Open returns an error message.

Set(String variable, String value)

The Set method sets the value of variable to value. If variable is invalid then variable will be added to the list of environment variables. If value is null then the variable will be removed. If variable and value are null then Set will return a listing of the variables and their values. If variable is null then Set will return an error message.

Alias(String name, String value)

The Alias method creates an alias for a command, name. If name is null or is already an alias Alias returns an error message. If value is null Alias returns an error message. If value and name are null Alias returns a list of the current aliases and their values.

Unalias(String name)

The Unalias method removes alias name from the alias table. If name is null or is not in the alias table Alias returns an error message.

Use Cases

- call ChangeDirectory with a relative path
- call ChangeDirectory with an absolute path
- call ChangeDirectory with a null path
- call the Up method
- call the Back method
- call the Forward method
- call ListDirectory with a relative path
- call ListDirectory with an absolute path
- call ListDirectory with a null path
- call Info with a relative path
- call Info with an absolute path
- call Info with a null path
- call Open with a relative path
- call Open with an absolute path
- call Open with a null path
- call Set with an existing variable and a value
- call Set with a non-existent variable and a value
- call Set with a non-existent variable and a null value

- call Set with an existing variable and a null value
- call Set with a null variable and null value
- call Alias with a name and a value
- call Alias with an already aliased name
- call Alias with a name with a !\$ in the value
- call Unalias with an alias
- call Unalias with a non-existent alias
- call Unalias with a null alias

Scenarios

Same as those in the console class.

CRC

Class Name:

Command

Responsibilities:

Execute the commands passed in from the Console class

Return the output of the commands

Collaborates With Classes:

Console

Control

MRV

Class Console

Black Box Specification

The Console class provides all textual I/O and navigation for the Web Shell product.

The Console class provides a command line UNIX-like console to the user. Through the console a user can execute the commands detailed in the Web Shell specification document version 1.1. Some commands provide feedback to the user via text in the terminal window. The Console class also provides history features to the user. By pressing the up or down arrows the user can navigate through the history list of commands. Upon finding the command of her choice the user can then modify and/or execute the command. The Console class also provides a number of environment variables to the user. These variables, detailed in the Web Shell specification document version 1.1, modify the output of commands and control the length of the history list.

Input–Output

The following is a list of commands available to the user supported by the Web Shells console. The console is case–sensitive, thus the commands should be typed, as shown, in lowercase. Words in brackets, [], indicate what sort of arguments the command expects. Arguments should not contain the brackets, []. The following list emphasizes the input to the commands, thus its explanation of command usage is somewhat abbreviated. Please consult the Web Shell requirements document version 1.1 for a more detailed explanation of the commands operations.

cd [name]

The change directory command takes one argument, the name of the directory to switch too. The directory name may be an absolute or relative path. Upon successful completion of the change directory operation the new absolute path with be printed in the console.

up

The up command changes the current directory to the current directory's parent. Upon successful completion of the up operation the new absolute path with be printed in the console.

back

The back command changes the current directory to the previous current directory as determined by the MRV–queue. Upon successful completion of the back operation the new absolute path with be printed in the console.

forward

The forward command changes the current directory to the next current directory as determined by the MRV–queue. Upon successful completion of the forward operation the new absolute path with be printed in the console. If the forward operation could not be completed an error message will be printed in the console.

ls [name]

The list directory command lists the contents of directory [name] in the console. [name] may be a relative or absolute path. If [name] is a plain file then [name] is listed. If [name] is omitted then the contents of the current directory are listed. The listing will denote the different types of files, directories and plain files.

info [name]

The info command lists the attributes of the file [name] in the console.

open [name]

The open command launches an external helper application to view the file [name].

set [variable [value]]

The set command allows the user to change the value of the environment variables. [variable] cannot contain white space. [value] can be enclosed in quotes but

they are not part of [value]. If [variable] and [value] are not defined then the set command lists the environment variables and their values in the console.

alias [name [value]]

The alias command allows the user to set up aliases for a command or a command sequence. [name] cannot contain white space. [value] can be enclosed in quotes but they are not part of [value]. If [name] and [value] are not defined then the alias command lists the aliases and their values in the console.

unalias [name]

The unalias command allows the user to remove aliases from the alias table.

Use Cases

- cd to a relative path
- cd to an absolute path
- cd to a null path
- use the up command
- use the back command
- use the forward command
- ls a relative path
- ls an absolute path
- ls with a null path
- info a relative path
- info an absolute path
- info a null path
- open a relative path
- open an absolute path
- open a null path
- set an existing variable to a value
- create a new variable
- create a new variable and set its value
- remove an existing variable
- list the existing variables and their values
- alias a name to a value
- alias an already aliased name
- alias a name with a !\$ in the value

- unalias an alias
- unalias a non-existent alias
- unalias a null alias

Scenarios

- Execute a number of commands, correctly and incorrectly. Use the history list. Execute a command from the history list. Use the history list again. Execute new commands.
- Execute a number of commands, including the cd command. Use the back and forward commands to change directories. Change directories using the cd command. Use the back and forward commands again.
- Execute a number of commands, including the cd command. Use the up command to change directories. Change directories with the cd command. Use the up command again.
- Alias a command. Use the alias. Alias a command with the !\$ symbol. Use the alias. Alias several commands concatenated with the ;. Use the alias. Alias several commands with the !\$ symbol concatenated with the ;. Use the alias.

CRC

Class Name:

Console

Responsibilities:

Interpret the users commands

Print command output and error messages to the terminal.

Collaborates With Classes:

Command

Control

History

Class History

Black Box Specification

The History class provides a command history list to the Web Shell console.

The History class provides a UNIX-like command history list to the Web Shell console. Through the console a user can browse through the history list using the up and down arrows. Upon selection of a previous command a user has the option to edit the command or simply execute it "as is".

Input–Output

The following is a list of public methods in the History class.

Previous

The Previous method returns the previous command in the history list. If the user is at the end of the the history list it returns the last entry in the list. If the list is empty it returns nothing.

Next

The Next method returns the next command in the history list. If the user is at the beginning of the the history list it returns the nothing. If the list is empty it returns nothing.

Add(String "command")

The Add method adds "command" to the history list. If "command" is null Add returns an error message.

SetLength

The SetLength method sets the number of commands that will be stored in the history list. If the new length is shorter than the old length the oldest commands will be truncated from the list.

Use Cases

- Move up in the list
- Move down in the list
- Set the length
- Set the length with a negative number
- Set the length with a very large number
- Add items to the list
- Add a null item to the list

Scenarios

- Move up and down in the list. Then add more items to list. Continue moving up and down in the list.
- Move up to the top of the list.
- Move down to the bottom of the list.

CRC

Class Name:

History

Responsibilities:

Keep track of history list

Provide the means to navigate the list

Provide the means to add items to the list

Collaborates With Classes:

Console

Class MRV

Black Box Specification

The MRV class provides the Most-Recently-Visited List queue used by the back and forward commands.

The Most Recently Visited queue contains a history of the recently visited directories. The queue has a fixed length determined by the environment variable MRVSIZE.

Input-Output

The following is a list of public methods in the MRV class.

Back

The Back method changes the current directory to be the previous directory and returns the new absolute path. If there is no previous directory then Back returns an error message.

Forward

The Forward method changes the current directory to be the next directory and returns the new absolute path. If there is no next directory then Forward returns an error message.

Add(String "directory")

The Add method adds "directory" to the MRV queue. If "directory" is null Add returns an error message.

SetLength

The SetLength method sets the number of directories that will be stored in the MRV queue. If the new length is shorter than the old length the oldest directory entries will be truncated from the list.

Use Cases

- Move back in the queue
- Move forward in the queue
- Set the length

- Set the length with a negative number
- Set the length with a very large number
- Add items to the queue
- Add a null item to the queue

Scenarios

Move forward and backward in the queue. Add more items to the queue. Continue moving in the queue.

Move forward to the beginning of the queue.

Move backward to the end of the queue.

CRC

Class Name:

Command

Responsibilities:

Keep track of the Most Recently Visited queue

Provide the means to navigate the queue

Provide the means to add items to the queue

Collaborates With Classes:

Command

Control

Class Sloth

The Sloth Class is the main class that creates an instance of the GUI class and an instance of the Console class.

Black Box Specification

Sloth starts the Web Shell with an optional argument and creates an instance of the GUI class and an instance of the Console panel. Sloth creates a window listener for allowing the user to move to a different panel. Sloth can terminate the Web Shell.

Input–Output

Upon starting the application, Sloth will take an argument and pass the argument to the Control class. It will also display the GUI instance mentioned in the Black Box Specification.

Acceptance Criteria

Sloth must be able to display a GUI class instance and a Console class instance for the user. It must provide the user a window listener that allows the user to select which view is active. It must be able to accept an argument upon launch of the application. Sloth must be able to terminate the application.

Use Cases

- Launch Sloth with no argument.
- Launch Sloth with a file as an argument.
- Launch Sloth with a directory as an argument.
- Launch Sloth with an http address as an argument.
- Launch Sloth with an invalid file or directory name.
- Select the Tree view.
- Select the Table view.
- Select the Command Line view.
- Terminate the application using the drop down menu.
- Terminate the application using the X in the window

Scenarios

- Launch Sloth with no argument: A window that contains a GUI instance and a Console instance. The user can select either of the views to make it the active window.
- Launch Sloth with a file as an argument: A window that contains a GUI instance and a Console instance. The user can select either of the views to make it the active window. Sloth will pass the input argument to the Control class.
- Launch Sloth with a directory as an argument: A window that contains a GUI instance and a Console instance. The user can select either of the views to make it the active window. Sloth will pass the input argument to the Control class.
- Launch Sloth with an http address: A window that contains a GUI instance and a Console instance. The user can select either of the views to make it the active window. Sloth will pass the input argument to the Control class.
- Launch Sloth with an invalid file or directory: A window that contains a GUI instance and a Console instance. The user can select either of the views to make it the active window. Sloth will pass the input argument to the Control class.
- Terminate the application with Close: When the drop down menu is selected, one of the user choices is Close. When Close is selected, Sloth terminates the application.
- Terminate the application with X in the window: When the user selects the X in the upper right hand corner of the window, Sloth terminates the application.

CRC Card

Class Name:

Sloth

Responsibilities:

Creates an instance of the GUI class and an instance of the Console Class.

Provides a window listener to make active any of the three views.

Can take an argument upon startup and passes it to the Control class.

Terminates the application.

Collaborates with the following classes:

Control

GUI

Console

Class TreePanel

The TreePanel Class displays files, directories and http addresses graphically as a tree.

Black Box Specification

TreePanel receives a file or directory as input and displays the input as a tree with the input as the root. A node in the tree can be expanded if it is a directory or an http address with links. A node can be closed if already expanded.

Input-Output

TreePanel receives a command to display a tree in its own panel. The input from the command can be files, directories and http addresses and TreePanel will create a graphical tree. It can also receive a command to expand a node in the tree. If a node is selected by clicking on the node with the mouse, the node is expanded and this information is passed to the Control class in order to update other views. If a node that represents a file is selected, no expansion of the node is possible.

Acceptance Criteria

TreePanel must be able to display a tree given a command to do so. It must be able to display the tree according to the file type associated with the command. It must be able to expand nodes that represent directories and http addresses with links. It must be able to close expanded nodes. When the tree is manipulated by the user, the closing or expansion of a node is information that is passed to the Control class. It must be able to handle invalid input such as null passed as a file or directory.

Use Cases

- Display a file as a root or node in a tree.
- Display a directory as a root or node in a tree.
- Display an http address as a root or node in a tree.
- Do nothing if a file is clicked on as a root or node in the tree.
- Expand a directory node if clicked on.
- Expand an http address node if clicked on and has links.
- Do nothing if a null file is given with an input command.

Scenarios

- Display Tree: When a command to display a tree is given, use the input to graphically display the given file and directory structure.
- Expand a node: When the user clicks on an expandable node, display the given file and directory structure.
- Close a node: When the user clicks on an expanded node, close the node to hide its file and directory structure.
- Pass null as a file or directory: When a command is received to update or display a tree and the input null, do nothing.

CRC Cards

Class Name:

TreePanel

Responsibilities:

Displays graphical tree.

TreePanel allows for the expansion of nodes.

(when they represent a directory or http address with links)

If nodes are expanded, the Control class is notified.

If an invalid file or directory is entered, an exception should be raised.

Collaborates with the following classes:

GUI

Control

Class TablePanel

The TablePanel Class lists the contents of directories and http addresses with links in a graphical table.

Black Box Specification

TablePanel receives a directory or http address as input and displays the contents in a table. The table can be sorted according to the listed attributes. If a folder or file is clicked on, Control is notified.

Input–Output

TablePanel receives a command to display a list in its own panel. The input from the command can be files, directories and http addresses and TablePanel will create a graphical table. It can also sort the table according to the attributes (column titles.) If a folder or file is clicked on, Control is notified.

Acceptance Criteria

TablePanel must be able to display a list of files given a command to do so. It must be able to sort the files according to attributes. It must be able to notify Control when a file or folder is clicked upon. It must be able to handle invalid input such as null passed as a file or directory.

Use Cases

- Display a file as list.
- Display the contents of a directory as list of files and directories.
- Display an http address as a list of links and directories.
- Click on a file.
- Click on a folder.
- Sort table according to user chosen attributes.
- Do nothing if a null file is given with an input command.

Scenarios

- Display List: When a command to display a list is given, use the input to graphically list the given file and directory structure.
- Sort the list: When the user clicks on an attribute of a file, the table will be sorted accordingly.
- Click on a file or folder: When a folder or a file is clicked upon, the Control instance must be notified.
- Pass null as a file or directory: When a command is received to update or display a list and the input null, do nothing.

CRC Cards

Class Name:

TablePanel

Responsibilities:

Displays graphical list.

Allows for the sorting of files and directories.

(according to the attribute selected by the user)

If a file or folder is clicked upon, notify the Control instance.

If an invalid file or directory is entered, an exception should be raised.

Collaborates with the following classes:

GUI

Control

Class GUI

The GUI Class creates instances of TreePanel and TablePanel.

Black Box Specification

GUI receives commands to create TreePanel and TablePanel from the Sloth class. It also passes inputs from Control to TreePanel and TablePanel. It also passes inputs from TreePanel and TablePanel on to Control.

Input–Output

GUI passes inputs from Control to TreePanel and TablePanel. It also passes inputs from TreePanel and TablePanel on to Control.

Acceptance Criteria

GUI class must be able to pass data and commands, in both directions, to Control, TreePanel and TablePanel. It also must be able to create TreePanels and TablePanels when commanded to do so from the Sloth class.

Use Cases

- Create a TreePanel.
- Create a TablePanel.
- Pass data and commands from Control to TreePanel.
- Pass data and commands from Control to TablePanel.
- Pass data and commands from TreePanel to Control.
- Pass data and commands from TablePanel to Control.

Scenarios

- Create a TreePanel: When Sloth commands the creation of a TreePanel, one will be

created.

- Create a TablePanel: When Sloth commands the creation of a TablePanel, one will be created.
- Pass data and commands from Control to TreePanel: When Control wants to pass data or commands to a TreePanel, GUI will pass these to TreePanel.
- Pass data and commands from Control to TablePanel: When Control wants to pass data or commands to a TablePanel, GUI will pass these to TablePanel.
- Pass data and commands from TreePanel to Control: When TreePanel wants to pass data or commands to a Control, GUI will pass these to Control.
- Pass data and commands from TablePanel to Control: When TablePanel wants to pass data or commands to a Control, GUI will pass these to Control.

CRC Cards

Class Name:

GUI

Responsibilities:

Create instances of TreePanel and TablePanel when Sloth dictates.

Pass data and commands to and from these panels and Control.

Collaborates with the following classes:

Sloth

TreePanel

TablePanel

Control

Class Control

Black Box Specification

Part of the Model–View–Controller pattern, the Control class coordinates the actions and data updates among the views and the file system.

The Control class manages the directory changes, switching from the local to the web file system and back. It does so transparently, without the need for the views to know about the filesystem, or the filesystem object to know about the views. The Control class also handles the GUI modes, as well as providing the other classes with debugging output capabilities. The Control class completely defines how the product interacts with the user, how the interface controls respond to user action, and how data updates are performed. The Control class is the one that turns the GUI on and off.

Input

- the GUI mode (as set by the console, and possibly by other views)
- the sorting mode (as set by the console and/or the GUI)
- the new directory the user wants to change to (from console and/or GUI)
- the file the user wants to open in an external viewer

Output

- refresh messages to all of the views
- the file system object that contains information about the current directory/web site
- show/hide the GUI view

I/O Relationship

- when the GUI mode is changed, it performs the necessary show/hide action on the GUI view
- when the sorting mode is changes, it refreshes the views
- when the directory changes, the views are refreshed and the file system object is updated / changes appropriately
- when a file is open, the Control class spawns the external program and hands it the file (downloaded a priori, if necessary)

Use Cases

- change directory
- change directory to an inexistent file/directory
- change directory from a file system to another file system
- run with no views
- set the GUI mode
- set the sorting mode
- open a file that has a handler application defined
- open a file that does not have a handler application defined

Scenarios

- change directory to a web site, retrieve the file system object, then open a file
- change directory to the local file system, retrieve the file system object, then open a file
- change directory to a web site, retrieve the file system object, then change directory to the local file system

CRC Card

Class name:

Control

Responsibilities:

refresh views
manage GUI mode
manage sorting mode
update file system object
open files

Collaborators:

FileSystem
GenericURL
Open
View
FileInfo

Class Open

Black Box Specification

It handles the opening of data viewers external to the WebShell.

The Open class spawns an external application as configured in the MIME type data file. It read that file, and creates a map of file types and associated handler applications. The way it executes the viewers is done in a OS-dependent manner (i.e. Netscape will be executed with different arguments depending whether we are in Windows or in Unix).

Input

- the MIME database
- the URL/file name to open

Output

- exceptions

I/O Relationships

- it throws an exception if the MIME database cannot be loaded, if the program cannot be run, or if the program returns an error

Use Cases

- open a file that has a handler applications defined
- open a file that does not have a handler application defined

Scenarios

- create a correct database, start the WebShell, open a file
- create an incorrect databases, start the WebShell, open a file
- delete the MIME database, start the WebShell, open a file

CRC Card

Class name:

Open

Responsibilities:

load the MIME database

validate the MIME database

open an external application and hand it the URL/file name

Collaborators:

Control

Class FileSystem

Black Box Specification

The FileSystem type provides an interface that different file systems (such as the local file system, and the web file system) must implement.

The FileSystem interface specifies the minimal functionality needed to access remote file systems. It provides for file names, links, file types, file sizes, as well as other file information. A file system is a read-only resource, a "mirror" of the live remote file system.

Input

- a URL the provides the root of the file system
- a directory in the current tree rooted at the given URL
- a file name

Output

- a list of files in the directory
- a list of attributes for the file

- local copy of the file

I/O Relationships

- when a directory listing is given, a list of file information entries is returned
- when a file name is given, a list of the file attributes is returned
- when a file name is given, the file is downloaded to the local file system

Use Cases

- connect to a remote file system
- connect to an inexistent remote file system
- list an existing directory
- list an inexistent directory
- query an existing file
- query an inexistent file
- make a local copy of an existing file
- make a local copy of an inexistent file

Scenarios

- connect to a remote file system, list files
- connect to an inexistent file system, list files
- connect to a remote file system, query inexistent file

CRC Card

Class name:

FileSystem

Responsibilities:

manage information extracted from the file system
(rooted at the given URL)

Collaborators:

Control

GenericURL

FileInfo

Class HTTPfs

Black Box Specification

HTTPfs is the web-based file system.

HTTPfs implements the FileSystem interface.

Input and Output

same as for the FileSystem interface

Use Cases

same as for the FileSystem interface

Scenarios

same as for the FileSystem interface

CRC Card

Class name:

HTTPfs

Responsibilities:

same as the FileSystem interface

Collaborators:

same as the FileSystem interface

HTMLParser

Class Localfs

Black Box Specification

Localfs is the web-based file system.

Localfs implements the FileSystem interface.

Input and Output

same as for the FileSystem interface

Use Cases

same as for the FileSystem interface

Scenarios

same as for the FileSystem interface

CRC Card

Class name:

Localfs

Responsibilities:

same as for the FileSystem interface

Collaborators:

same as for the FileSystem interface

Class FileInfo

Black Box Specification

This class organizes and encapsulates information about a single file.

FileInfo contains information on the file such as file name, file size, file type, and other attributes available.

Input

- none

Output

- none

I/O Relationships

- none

Use Cases

- none

Scenarios

- none

CRC Card

Class name:

FileInfo

Responsibilities:

store file attributes

Collaborators:

none

Class GenericURL

Black Box Specification

The GenericURL class implements URL handling policy given by the IETF RFCs 1738 and 1808.

A GenericURL can be modified using relative or absolute paths, and these updates can change its meaning from web-based to local and vice-versa.

Input

- an absolute or relative path given as a string

Output

- a file system corresponding to that URL

I/O Relationships

- when the URL is updated and the file system is queried, the GenericURL creates a new file system object appropriate to the new URL

Use Cases

- create a GenericURL for a web site
- create a GenericURL for a local directory
- update the GenericURL with a legal relative path
- update the GenericURL with an illegal relative path
- update the GenericURL with an absolute path
- retrieve the file system when the URL is set
- retrieve the file system when the URL is not set

Scenarios

- create a GenericURL, update it with a relative path
- create a GenericURL, update it with an absolute path
- create a GenericURL, update it with an absolute path, query file system

CRC Card

Class name:

GenericURL

Responsibilities:

manage URLs as specified in RFCs 1738 and 1808

allow relative and absolute updates
create file systems appropriate to the URL description

Collaborators:

HTTPfs

Localfs

Class HTMLParser

Black Box Specification

The HTMLParser extracts the directory information from the HTML code of a web page, as defined in the WebShell requirements document v1.1.

The HTMLParser extracts the lexemes that compose the URL references in a web page, and returns them one by one, for the WebShell to use in the displayed information. The HTMLParser is a simple, non-reentrant parser, capable of extracting the URLs efficiently.

Input

- a stream of characters, representing an HTML page

Output

- a URL stream, one per line

I/O Relationships

- the URL stream must contain all and only the URLs from the input HTML data

Use Cases

- provide as input HTML pages of various length
- provide as input some non-HTML data

Scenarios

- due to limited capabilities, the Scenarios are identical to the Use Cases

CRC Card

Class name:

HTMLParser

Responsibilities:

extract all the URLs from input HTML stream

Collaborators:

none

Class View

Black Box Specification

The View class is an abstract class (an interface). It provides a standard interface, the same way the FileSystem interface does.

The View interface defines the functionality available in a View, specifically functions to refresh the display, and functions to identify the view.

Input

- the refresh event
- a request for the name of the view

Output

- the name of the view

I/O Relationships

- the name of the view must be unique inside the system. More specifically, the GUI class must always return the string "GUI".

Use Cases

- get the View name
- refresh

Scenarios

- same as above

CRC Card

Class name:

View

Responsibilities:

refresh the view

provide the name of the view

Collaborators:

Control

FileSystem