SmallWiki
Collaborative Content Management

Lukas Renggli, renggli@iam.unibe.ch
Software Composition Group
University of Bern, Switzerland

October 2003

Abstract

A Wiki is a collaborative software to do content management. Although there are a lot of different Wiki implementations available today, they all lack the possibility to be extended and to adapt to the needs of their users. SmallWiki is a new and fully object-oriented Wiki framework in Smalltalk, that has got a lot of unit-tests included. This documentation gives an overview how to run it, about its design and implementation, and provides a few examples on writing extensions.

Acknowledgments

I am grateful to Stephane Ducasse and Roel Wuyts for all those useful discussions about the design and the implementation of SmallWiki. I would like to thank Alexandre Bergel for writing examples and an initial documentation. Thanks to all the people who have submitted suggestions, patches and bug reports.

Contents

Chapter 1

Introduction

1.1 Context

The term *Wiki* usually means the collaborative software used to create, edit and manage hypertext pages on the web. A Wiki enables the users to author their documents using a simple markup language within their preferred web-browser.

Translated from the Hawaiian language *Wiki wiki* means *fast* and that is exactly what the collaborative editing process of a WikiWiki Web is all about: Everybody should be able to create and update pages, without the need of user-name and password to login. However there are wiki vandals around that abuse the general public access and make it necessary to protect the content with security mechanism.

1.2 Problem

There are more than 150 Wiki implementations available [5] and most of them are open source. There are even a few implementations available written in different Smalltalk dialects, so why did we create a new one?

All the implementations we had a look at have major flaws in extensibility: they don't provide a proper object-oriented design that is covered by unit tests. Moreover they all keep the content of the pages within strings, what makes it painful to render and search the wiki. These wikis haven't been designed for extensibility!

The existing Smalltalk wikis are old and it seems that the developers don't want to touch their running systems. Both, WikiWorks and SqueakWiki, have some of their domain code within external files, which makes the source hard to understand as it is not possible to use the editing and debugging facilities of the Smalltalk environment.

Figure 1.1: SmallWiki in the Web-Browser

1.3 Solution

As stated in the previous section, all the current wiki implementations have problems with extensibility and the design. We didn't want to make the same mistakes and put together the following basic requirements at the very beginning of the project:

Object-Oriented Design. SmallWiki provides an object oriented domain model. As an example, the content of the pages is parsed and stored as a tree of different entities representing text, links, tables, lists, etc.

Extensibility. Everything in SmallWiki can be extended: page types, storage mechanism, actions, security mechanism, web-server, etc. Plug-ins can be shared within the community and loaded independently of each other into the system.

Open Source. SmallWiki is released under the MIT license which grants unrestricted rights to copy, modify, and redistribute as long as the original copyright and license terms are retained.

Test Suites. SmallWiki is heavily tested. There are more than 200 unit tests included with the core of SmallWiki. This makes it easy to change and verify the code and comes in extremely useful when porting SmallWiki to other Smalltalk dialects or when writing extensions.

1.4 Overview

Chapter 2: SmallWiki in a Nutshell will give you all the needed information to download and run SmallWiki on your computer.

Chapter 3: SmallWiki Design will provide documentation about the most important classes, their responsibilities, and their use. This part of the documentation is partially generated automatically from the class- and message-comments of SmallWiki.

Chapter 4: Extending SmallWiki will give you several tutorials on building plug-ins for SmallWiki.

Chapter 2

SmallWiki in a Nutshell

SmallWiki has been implemented using VisualWorks 7, which can be downloaded for free from the Cincom home-page [8]. To download the latest implementation of SmallWiki itself use Cincom Public StORE or you can have a look at the goodie directory within a current VisualWorks distribution.

2.1 Loading Into the Image

1. Load the bundle named *SmallWiki* from Cincom Public StORE or from the goodies directory into your image. You will be asked to load some additional bundles like the SIXX XML serialisation library [7], the Swazoo web-server [6] and the SmaCC parser generator [3].

2. If you like to have some example extensions available, also load the bundle called *SmallWiki Examples*.

2.2 Running the Tests

1. Make sure you have the package *RBSUnitExtensions* installed.

2. Locate the package *SmallWiki Tests*, that contains all the SUnit tests of SmallWiki.

3. Select all the test-cases and click on run.

4. You should see a green bar, if all tests pass.

2.3 Starting a Server

1. While loading SmallWiki a workspace should have opened automatically with useful commands to run the web-server. Read trough the whole text to see some more possibilities for configuration.

2. If you are in a hurry, just evaluate the first expression

   ```
   server := SmallWiki.SwazooServer startOn:  8080
   ```

   which starts the server on the port 8080.

3. Switch to your favourite web-browser and point it to

   ```
   http://localhost:8080
   ```

4. Have a look at the *Information* folder and read through those pages for important news published there.

2.4 Editing a Page

The syntax used to edit a SmallWiki page is simple and easy to remember.

Paragraphs. As carriage returns are preserved, simply add a newline to begin a new paragraph.

Headers. A line starting with `!`s becomes a header line.

Horizontal Line. A line starting with _ (underscore) becomes a horizontal line. This is often used to separate topics.

Lists. Using lines starting with `#`s and `-`s, creates a list: A block of lines, where each line starts with `-` is transformed into a bulleted list, where each line is an entry. A block of lines, where each line starts with `#` is transformed into an ordered list, where each line is an entry.

Tables. To create a table, start off the lines with `|` and separate the elements with `|`s. Each new line represents a new row of the table.

Pre-formatted. To create a pre-formatted section, begin each line with `=`. A pre-formatted section uses equally spaced text so that spacing is preserved.

Links. To create a link, put it between `*`s. There are three different types of links:

Internal Link. If the item exists in the Smallwiki (e.g. `*Title of Item*`), a link to that item shows up when the page is saved. In case the item does not already exist, the link shows up with a create-button next to it; click on it to create the new item.

External Link. If the link is a valid url (e.g. `*http://www.google.ch*`), a link to that external page shows up.

Mail Link. If the link is an e-mail address (e.g. `*self@mail.me.com*`), a link to mail that person shows up, but it is obfuscated to prevent robots from collecting.

You can also alias all these links using `>`. So, you can create a link like this: `*Alias>Reference*`. The link will show up as Alias, but link to Reference. For images, the alias text will become the alternate text for the image.

HTML. Use any HTML anywhere you want. Some useful HTML tags are:

To make something bold, surround it by `<b>` and `</b>`.

To make something italic, surround it by `<i>` and `</i>`.

To make something underlined, surround it by `<u>` and `</u>`.

The table 2.1 lists all the mark-up tags and compares them to the syntax of SqueakWiki and WikiWorks.

	SmallWiki	SqueakWiki	WikiWorks
Heading	`!`, `!!`, ...	`!`, `!!`, ...	`!`, `!!`, ...
Horizontal Rule	_	_	`-`
Numbered List	`#`	`#`	`#`
Bullet List	`-`	`-`	`*`
Table	`|`	`|`	{, |, }
Pre-formatted	`=`	`=`	`<pre>`
Link	`*Reference*`	`*Reference*`	`[Reference]`
Link Alias	`*Alias>Reference*`	`*Alias>Reference*`	`[Alias>Reference]`
Smalltalk Code	`[Code]`	n/a	n/a

Table 2.1: SmallWiki Syntax compared to SqueakWiki and WikiWorks

2.5 Accessing the Admin Account

Some commands – as changing the design, removing pages or modifying the history – might require you to log-in. A default administrator has been created during installation with the

user-name `admin` and the password `smallwiki`. You should consider changing these default settings using the provided workspace. Note that there is more advanced user-interface underway to manage all the security aspects of SmallWiki.

Chapter 3

SmallWiki Design

The most widely used design patterns used in SmallWiki are the Composite [1, page 137] and the Visitor [1, page 371] patterns.

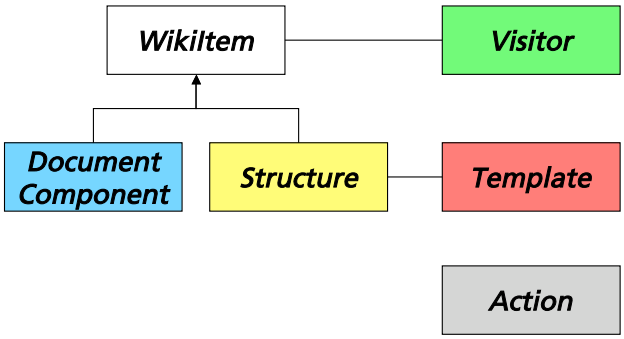

Figure 3.1: Core Design

All the classes seen in Figure 3.1 are abstract. Their concrete subclasses will be dis-

cussed in the following subsections. The subclasses of `WikiItem` represent the model in the MVC paradigm and might be visited using subclasses of the `Visitor` hierarchy. As all the rendering is done within different visitors, this part can be seen as the view. At last we have the controller, represented by the hierarchy below the `Action` class. Actions are used to do modifications on the model and to start the different visitors to generate the appropriate views.

3.1 Server

The basic serving is done with the chain-of-responsibilities design pattern [1, page 225] in the serving protocol. Incoming requests are passed to the first possible candidate that is able to handle it. The request is analysed and processed within this structure and if necessary processed or passed to one of its children.

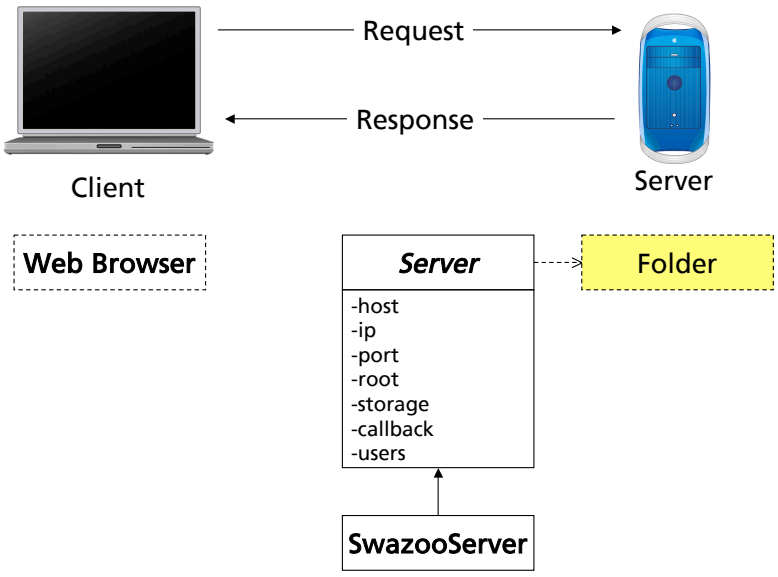

Figure 3.2: Server Setup

The server class has been designed to be subclassed and to provide a common interface to different server implementations. A server might get started using the messages `#start`, `#startOn:`, `#startOn:host:ip:` or by simply instantiating using the message `#new`, con-

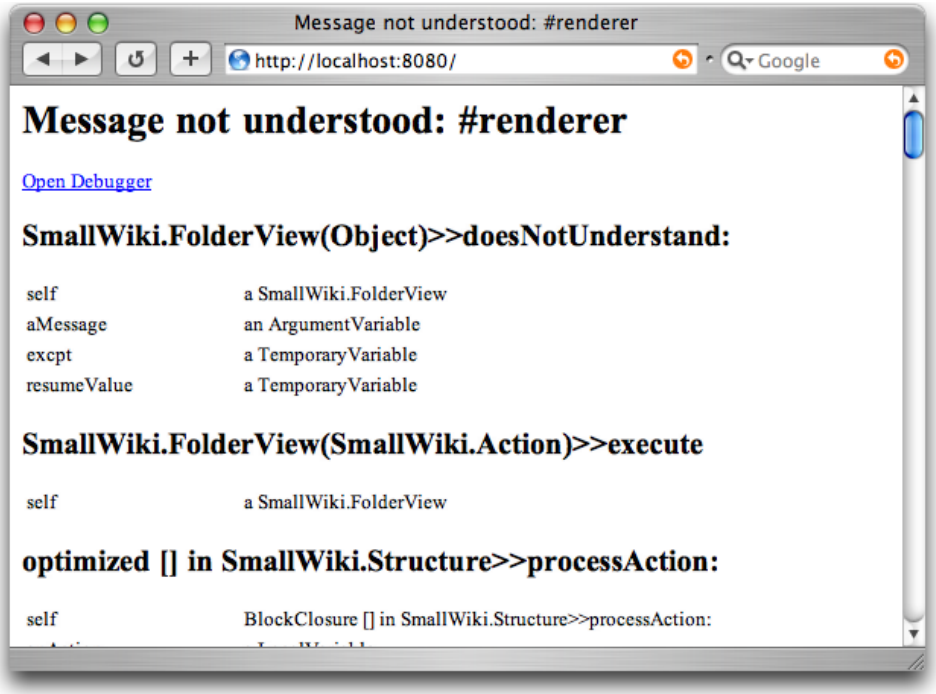

Figure 3.3: Stack Dump in the Web-Browser

figuring and starting manually. Please note that the server is not a singleton, so there might be multiple instances running within the same image.

```
server := SwazooServer startOn: 8080.
```

The instance variable `root` represents the root-entity of the wiki-tree, what is usually a folder. When starting a new server, a default configuration will be created. Take care that you don't accidentally call the write-accessor for the root on a running wiki, as all the subentries will be destroyed immediately without the possibility of going back. If you want to have a look at the model of your wiki, evaluate the following expression:

```
server root inspect
```

The default server has no automatic storage mechanism assigned; this is basically useful when developing for SmallWiki and saving the image manually. If you use the wiki in a production environment make sure that you assign a working storage-strategy and test extensively that it is fitting your needs. If you develop other storage strategies, please let us know as we are interested to integrate them into the main-distribution.

```
server storage: ImageStorage new.      " fast and secure persistence "
server storage: SIXXStorage new.       " slow dump-out using xml "
server storage: nil.                   " no persistence "
```

The responsibility to pass the request to the root node of the wiki is taken by the server. Also there will be caught all kinds of exceptions and being displayed as a stack-dump on the client side. The link *Open Debugger* can be used to open the debugger in VisualWorks within the context that caused the error and investigate the problem further.

accessing-users

- `Server>>userAdd: anUser`
  Add a new user to the receiver. Any user with the same username will be overridden.
- `Server>>userAt: aString`
  Return the user with aString as name, if there is no such user the default anonymous user is returned.
- `Server>>userAt: aString ifAbsent: anExceptionBlock`
  Return the user with aString as name, if there is no such user anExceptionBlock is evaluated.
- `Server>>userIncludes: aString`
  Return true if the receiver has got a user with the given username.
- `Server>>userRemove: aString`
  Remove the user with the given username from the receiver.

configuration

- `Server>>defaultRoot`
  Return the default wiki that will be used when setting up a new server.

serving

- `Server>>isServing`
  Return true if the receivers web-server is up and running.
- `Server>>restart`
  Restart the web-server of the receiver.
- `Server>>start`
  Start the web-server of the receiver.

- `Server>>stop`
  Stop the web-server of the receiver.

serving-private

- `Server>>process: aRequest`
  Start the chain of responsibilities in the root of the wiki. Any unhandled exceptions thrown while processing the request will be displayed as a stack-trace within the browser of the client.

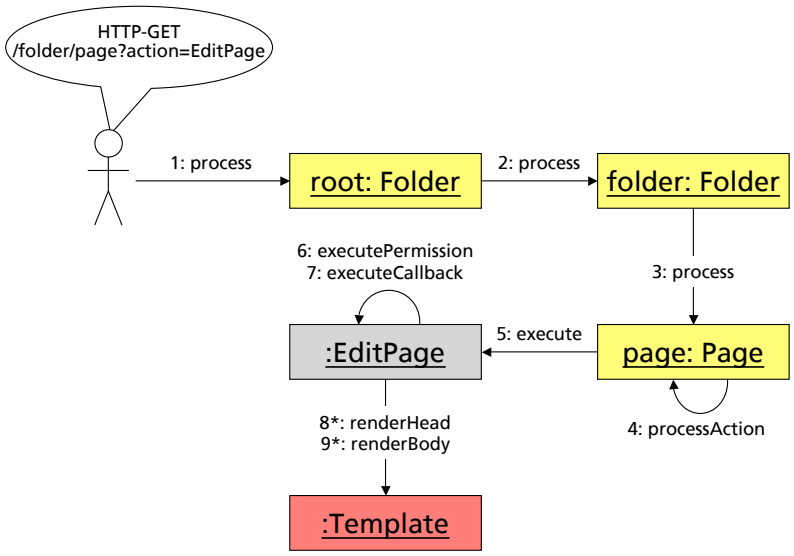

Figure 3.4: Chain of Responsibility: Content Serving

Imagine a user entering an URL such as

`http://www.smallwiki.org/folder/page?action=EditPage`

to edit the page *page* contained in the folder *folder*. The following steps, as seen in the collaboration diagram of Figure 3.4, are taken:

1. The web-server gets the request emitted by the client and starts the look-up process by delegating it to the root folder.

2. Because the target isn't the root chapter itself, the request is delegated to a the folder called *folder*.

3. As in the previous step, in this folder the request is delegated at the page called *page*.

4. There is no-one else that could be interested in this request, it is therefore processed by extracting the parameters and determining the action that should be executed. In this example the class `PageEdit` will be instantiated, initialised and the message `#execute` will be sent.

5. The action first checks the permissions of the user and evaluates the callbacks, see page 25 chapter 3.4 for further information.

6. Then the action asks all the template-components to emit their html-header and their html-body, see page 28 chapter 3.5 for further information.

3.2 Structure

The structure is the basic entity of SmallWiki, representing the model of a single page. A structure is identified by exactly one URL and is usually included in a composite-tree of other structures. The three concrete subclasses of `Structure` are: `Page` and `Resource` as components and the `Folder` as composite. In fact `Structure` should not only be the subclass of `WikiItem`, but also of `Model`. As the visiting aspect, however, is far more important, the messages provided in `Model` have been copied from this system class.

A structure provides basic navigational accessors to its parents, children and sisters in the wiki-tree. The basic serving is done with the chain-of-responsibilities design-pattern in the serving protocol. The resolving protocol provides messages to look-up other structure-items using their name.

All the structures have a title, a back-reference to their parent and might contain user defined properties, what is something like a dictionary containing symbols as key and any other objects as values. Structures are versioned automatically using a reference pointing to the previous version of the same page. Make sure to override the message `#postCopy` to make it work correctly.

accessing

- `Structure>>id`
  Return the id of the receiver, that is a string build from the title of the structure. The id is used to identify a structure within its parent and therefore has to be unique.

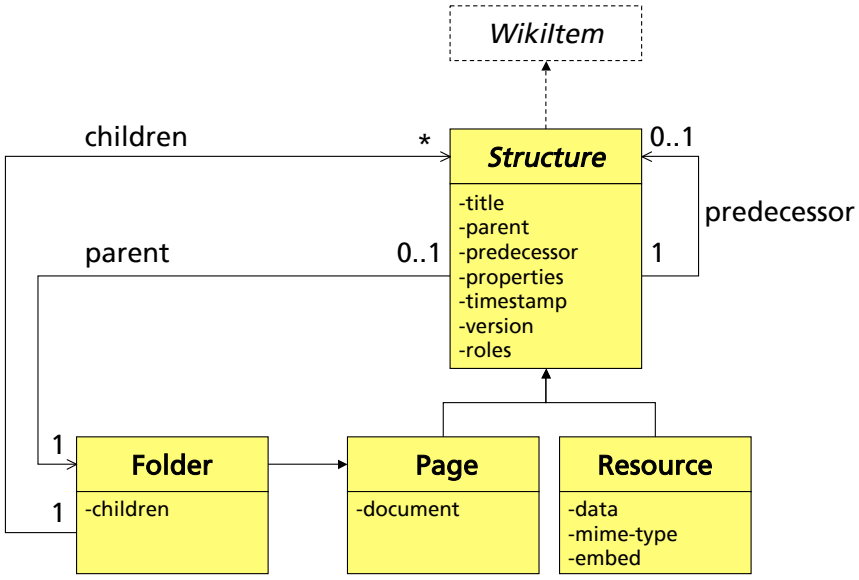

Figure 3.5: Structure Composite

- `Structure>>parent`
  Return the parent of the receiver. In the case the receiver is the root node nil is returned.

- `Structure>>predecessor`
  Return the previous version in the history of the receiver. If there is no history information available nil is returned.

- `Structure>>roles`
  Return a collection of roles which are applied when the receiver processes a query. All the roles of the current user are replaced with the corresponding ones returned by this message.

- `Structure>>timestamp`
  Return a timestamp with the latest modification-time of the receiver.

- `Structure>>title: aString`
  Set the title of the receiver. If the receiver is a child of another structure the title

is checked to be unique. In case of a problem, a `DuplicatedStructure` exception is thrown.

- `Structure>>version`
  Return the version-number of the receiver, the numbering starts with version 0.

accessing-calculated

- `Structure>>parents`
  Return an ordered collection with all the structures from the root up to and including the receiver of the message.

- `Structure>>root`
  Return the root node of the receiver.

- `Structure>>url`
  Return a unix-path string representing the URL of the receiver. The URL is unique within a wiki-tree and contains the id of the receiver and all its parents ids, except for the root.

- `Structure>>versions`
  Return an ordered-collection containing the receiver and all its older versions.

accessing-navigation

- `Structure>>first`
  Return the first node of the receiver's parent.

- `Structure>>last`
  Return the last node of the receiver's parent.

- `Structure>>next`
  Return the next node of the receiver's parent.

- `Structure>>previous`
  Return the previous node of the receiver's parent.

configuration

- `Structure>>defaultAddTarget`
  Return the target where to put new children. The default implementation returns the parent of the receiver, but structures that contain children usually want to return self.

copying

- `Structure>>postCopy`
  Copy a selection of the instance-variables and update the time-stamp of the receiver. Subclasses should override this message to do a deep-copy of their data and call super. This is the key-message to make the versioning mechanism work properly.

properties-inherited

- `Structure>>properties`
  Return a property manager including the values of all inherited properties. Changes to the returned object does not change the receivers property manager.

- `Structure>>propertyAt: aKey`
  Return the value of the property of the receiver with aKey. If there is no such property defined, the look-up is retried in the parent of the receiver.

- `Structure>>propertyAt: aKey ifAbsent: anExceptionBlock`
  Return the value of the property of the receiver with aKey. If there is no such property defined, the look-up is retried in the parent of the receiver. If no such property could be found, anExceptionBlock is evaluated.

- `Structure>>propertyAt: aKey put: aValue`
  Set the property aKey to aValue in the receiver. This message does the same as `#localPropertyAt:put:` and is here simply for conveniance.

properties-local

- `Structure>>localProperties`
  Return the properties of the receiver.

- `Structure>>localPropertyAt: aKey`
  Return the value of the property of the receiver with aKey. If there is no such property, nil is returned.

- `Structure>>localPropertyAt: aKey ifAbsent: anExceptionBlock`
  Return the value of the property of the receiver with aKey. If there is no such property, anExceptionBlock is evaluated.

- `Structure>>localPropertyAt: aKey put: aValue`
  Set the property aKey to aValue in the receiver.

resolving

- `Structure>>privateResolveChild: aString`
  As we usually don't have children, return nil. Subclasses with children might want to override this message.

- `Structure>>privateResolvePath: aCollection`
  Resolve a whole path starting at the receiver. If there is an error matching the path, nil is returned.

- `Structure>>resolveTo: aStringPath`
  Start the resolving-process at the receiver with the resolving-algorithm depending on the count of identifiers in aPathString:

  1. if the first character of the path is a separator the look-up is started in the root-node and processed downwards.
  2. if the first character is not a separator ...
     (a) and an empty path is given, the receiver is returned.
     (b) and a path with exactly one entry is given, a child with that name is looked for.
     (c) and a path with one or more entries is given, a look-up is started in the parent of the receiver.

  The first matching structure is returned or nil if no appropriate item could be located in the tree. To see a bunch of examples about the use of this message, have a look at the tests in protocol `resting-resolving` of the class `StructureTests`.

serving

- `Structure>>process: aRequest`
  Process a basic request. First the security information contained in the request is updated, then it is decided if the request should be handled by the current component or one of its children.

- `Structure>>processAction: anAction`
  Executes the action on myself and catch basic errors.

- `Structure>>processChild: aRequest`
  The default structure has got no children, therefore we process a not-found message.

- `Structure>>processSecurity: aRequest`
  Update the roles of the current user according to the current role configuration. See `#updateRoles:` in the `User` class for additional information.

- `Structure>>processSelf: aRequest`
  Look for an action that might be executed on the current structure. If there is no action given, the default one is executed.

testing

- `Structure>>isComposite`
  Return true if the structure has got the possiblity to hold children.

- `Structure>>isEmbedded`
  Return true if the structure should be embedded into documents referencing the receiver.

- `Structure>>isRoot`
  Answer whether the receiver is the root node. The root is a folder by default and the only structure having no parent in the wiki-tree.

- `Structure>>isSuccessor`
  Answer whether the receiver has got a previous version in the history.

versions

- `Structure>>nextVersion`
  Copy the receiver to be used in the history and return a the receiver.

- `Structure>>nextVersionBecome: aStructure`
  This message creates a copy of the receiver and puts aStructure into the history. References pointing to the receiver will be still valid, as the current version stays the same object all the time. Right now the new version must be of the same class than the receiver, else an exception is thrown.

- `Structure>>versionNumber: anInteger`
  Return the version anInteger of the receiver or nil if not present.

- `Structure>>versionRestore: anInteger`
  Restore the version anInteger. In other words: the version anInteger will become the current one, but all the other modifications are still kept in history.

- `Structure>>versionRevert: anInteger`
  Revert to the version anInteger in history. All the newer modifications will be lost.

- `Structure>>versionTruncate: anInteger`
  Truncate all the history information behind the version anInteger.

3.2.1 Folder

The folder groups a number of children. `Folder` is a subclass of `Page`, therefor they also contain a document that might be used to describe the contents.

accessing

- `Folder>>children`
  Return a collection of all the children of the receiver.

children-accessing

- `Folder>>at: aString`
  Return child of the receiver with id aString or nil if absent.
- `Folder>>at: aString ifAbsent: aBlock`
  Return child of the receiver with id aString or nil if absent.
- `Folder>>includes: aString`
  Return true if the receiver contains a child with the id aString.
- `Folder>>uniqueTitle: aString`
  Proposes an unique name for a child within the receiver using aString base name.

children-structure

- `Folder>>add: aStructure`
  Add aStructure as a child to the receiver. A DuplicatedStructure exception is raised in case there is already a child with the same name.
- `Folder>>copy: aStructure`
  This message is basically the same as `#add:` but it creates a copy of the structure and makes sure that the title is unique within the receiver before adding.
- `Folder>>remove: aStructure`
  Remove aStructure from the list of children of the receiver. In case there is no such child an exception is raised.

configuration

- `Folder>>defaultChildrenCollection`
  When changing this message, you should modify the following messages: `#at:ifAbsent:`, `#add:`, `#remove:` and `#children`.

- `Folder>>defaultDocument`
  Return the default document used when a new folder is created.

navigation

- `Folder>>next: aStructure`
  Return the next node of the receiver's child aStructure.
- `Folder>>previous: aStructure`
  Return the previous node of the receiver's child aStructure.

3.2.2 Page

A page is the most important and probably the most used class of the Structure hierarchy. As a sole entity it contains a composite of documents modeling the contents of the page that the user entered using the wiki-syntax. When initializing the instance a default document will be created to make the user aware of the newly created page.

accessing

- `Page>>document`
  Return the current document of the page.

configuration

- `Page>>defaultDocument`
  Return the default document used when a new page is created.
- `Page>>defaultVisitor`
  Return the default visitor used when rendering this document to html.

3.2.3 Resource

A resource might contain any data, like images, videos, sound, pdf or zip files. In fact it can be anything that you want to include within your pages or you want to provide as a possibility to download.

The mime-type of the data is used to determine how the given resource should be rendered. As an example images and videos should be displayed inside the html document, whereas zip-files are only references as a link to allow the user to download the file.

testing

- `Resource>>isApplication`
  Return true if the mimetype of the receiver is application-data. This message will match types like: application/octet-stream, application/oda, application/postscript, application/zip, application/pdf, etc.

- `Resource>>isAudio`
  Return true if the mimetype of the receiver is audio-data. This message will match types like: audio/basic, audio/tone, audio/mpeg, etc.

- `Resource>>isEmbedded`
  Return true if the resource of the receiver should be embedded into the desired context. Return false if the resource should be simply linked.

- `Resource>>isImage`
  Return true if the mimetype of the receiver is image-data. This message will match types like: image/jpeg, image/gif, image/png, image/tiff, etc.

- `Resource>>isText`
  Return true if the mimetype of the receiver is text-data. This message will match types like: text/plain, text/html, text/sgml, text/css, text/xml, text/richtext, etc.

- `Resource>>isVideo`
  Return true if the mimetype of the receiver is video-data. This message will match types like: video/mpeg, video/quicktime, etc.

3.3 Document

The document hierarchy describes the content of a wiki page. It includes all the basic elements to represent a text such as paragraph, table, list, links, etc. When the user enters a text using the wiki syntax it is parsed using SmaCC [3] and the abstract syntax tree is stored within the page.

As Table 2.1 shows, the syntax of SmallWiki is similar to SqueakWiki or WikiWorks. Changing the grammar of the parser is no big deal, if you are more familiar with a different one and want to support that. However, as for all other parsers, it is difficult to write extensions that can be added and removed independently in order to parse new document entities.

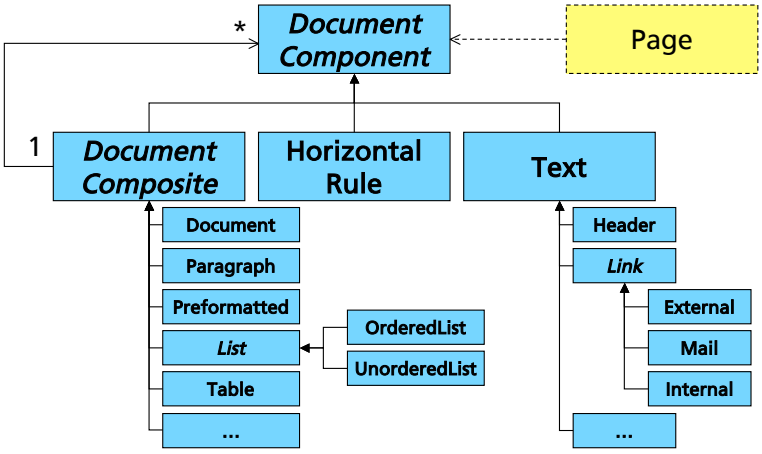

Figure 3.6: The Document Hierarchy

3.4 Action

Actions are instantiated by a structure and they are initialized with that structure and the current request using the constructor method `#request:structure:`. Actions have basically two tasks: performing the action itself and initiating or doing the GUI rendering. Actions do also represent the context in which a page is rendered as they know about their structure, the request, the response, and the security status.

Action Protocol

This part of the action is used to handle the requests. The message `#execute` is called by the structure after initializing the required instance variables. It checks the security permissions of the current user, evaluates the callbacks and starts the rendering by calling `#render` on itself. The running action might use the accessors to manipulate and mediate with the current environment. It is usually not necessary to override the message `#execute`, use the callback mechanism described in Chapter ?? on page ?? instead.

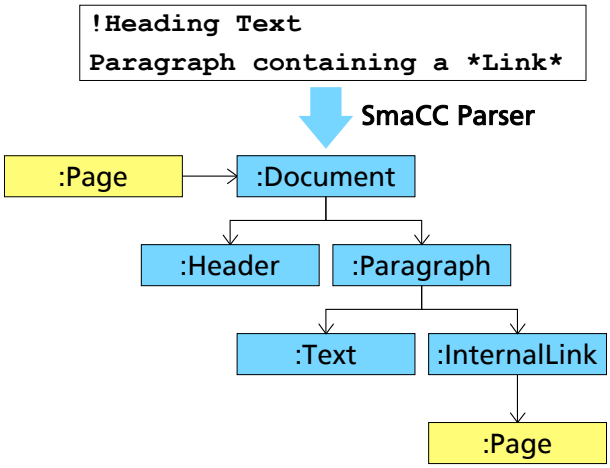

Figure 3.7: Parsing a Wiki Document

Rendering Protocol

The rendering process is started from the message `#render` at the end of `#execute`. The message `#render` fetches the collection of templates of the associated structure and starts generating the XHTML output. It asks each template to render the content they want to emit into the `<head>...</head>` part of the output. Afterwards the body part `<body>...</body>` is generated and again every template might contribute its content into that part. As explained on page 28 in chapter 3.5, there is always an instance of `TemplateBodyContent` available calling back the message `#renderContent` of the action: override this message to let your action render its user-interface. You should not change the state of any component inside the rendering protocol, as an action is unable to know when and how many times it is actually called.

accessing-heading

- `Action>>heading`
  Return the full heading of the receiver, containing the title of the receiver and the one

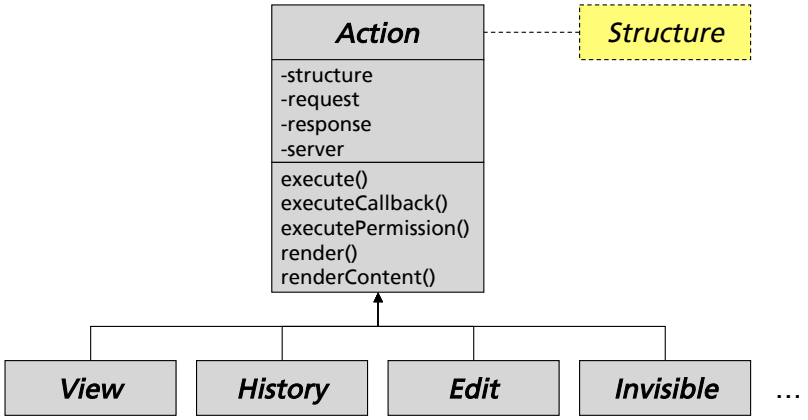

Figure 3.8: The Action Hierarchy

of the structure to be handled. Do not override this message, instead have a look at `#headingAction` and `#headingStructure.`

- `Action>>headingAction`
  Return the title of the reveiver to be used in the heading. Do override this message, if you want to provide something different. This might return nil, if it is not appropriate.

- `Action>>headingStructure`
  Return the title of the current structure to be used in the heading. Do override this message, if you want to provide something different. This might return nil, if it is not appropriate.

action

- `Action>>execute`
  Usually it is not necessary to override this message, instead use the provided callback mechanism. Still there are rare cases where you need to have full control over the execution process; but do not generate any output in here, use `#renderContent` instead.

The rendering is called automatically, if there hasn't been a redirect response created while checking the permissions or while executing the callbacks.

- `Action>>executeCallback`
  Override this message to provide your own way of evaluating callbacks. This implementation only executes the anchor-callback when there are no form-callbacks being executed. This prevents from accidently executing form and anchor callbacks at the same time, what is usually not intended.

- `Action>>executePermission`
  Override this message to check permission before anything inside this action is executed. By default an action might be used by all users, so no permissions are asserted.

rendering

- `Action>>render`
  This message starts the basic html rendering of a page. Unless you do not want the templates and the html addendum to be rendered, do not override this message.

- `Action>>renderContent`
  Override this message to customize the output of this action. Do not change the state of the component in this message.

security

- `Action>>assertPermission: aPermission`
  Assert the presence of a Permission in the current session. If no such permission is present an `UnauthorizedError` is thrown and an error page will be renderered instead of the one of the current action.

testing

- `Action>>isIndexable`
  Override this message and return true to tell search-engines to index the contents of this actions.

3.5 Template

Templates are used to render common parts of wiki-pages. They are defined within a collection hold in the root of the wiki and in combination with a selected Stylesheet, they provide the look-and-feel of the wiki. As the templates are hold in the property manager of the structure, they are shared within all childrens of a folder unless there is a new definition.

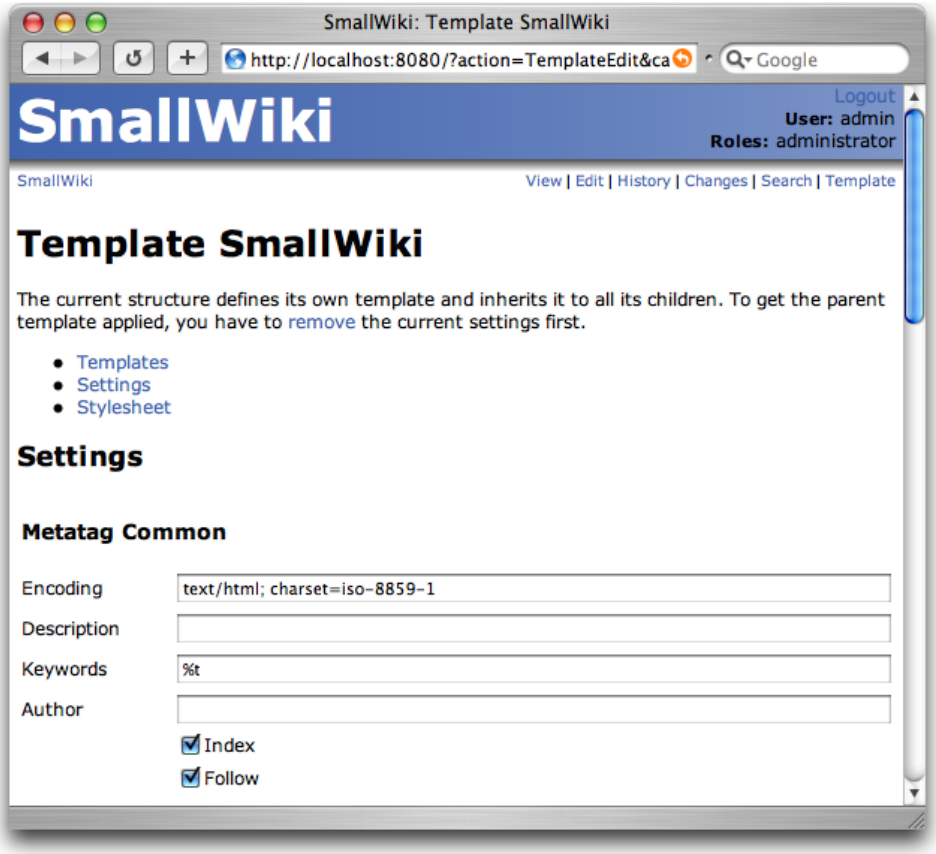

Figure 3.9: The Template Editor

private

- `Template>>expand: aString for: anAction`
  Expand aString within the context of anAction. This is often used to let the user specify dynamic parts within the settings of the templates. Currently the following tags are supported:
  - `%a` the title of the action
  - `%h` the host-name of the server
  - `%i` the ip-number of the server
  - `%l` the the url of the structure
  - `%m` the modification time of the structure

- `%p` the port-number of the server
- `%r` the title of the root structure
- `%t` the title of the structure
- `%u` the name of the current user
- `%v` the version-number of the structure

rendering

- `Template>>renderBodyWith: anAction on: html`
  This message is called when the action should render its content to the body-part of the resulting XHTML document. The default implementation is empty.

- `Template>>renderConfigWith: anAction on: html`
  This message is called when the configuration form of the receiver should be rendered. This message is solely called from the `TemplateEdit` action to let the user specify his settings. The default implementation is empty.

- `Template>>renderHeadWith: anAction on: html`
  This message is called when the action should render its content to the head-part of the resulting XHTML document. The default implementation is empty.

3.5.1 Head Template

The class `TemplateHead` should be used for templates only rendering to the header of the output-file. If you want to render to the head and to the body use `TemplateBody` as a superclass instead.

3.5.2 Body Template

The class `TemplateBody` should be subclassed in most of the cases to create a new template component. Don't forget to implement the message `#title` on the class-side to return a string describing this class. The title is also used by default to identify the associated CSS-id and to render it into the body part. The messages `#defaultId` and `#defaultTitle` might be used to change this behaviour. The user is always able to edit the id and title from within the template-editor in the web-browser to customise the template to his needs and to the applied stylesheet.

rendering

- `TemplateBody>>renderBodyWith: anAction on: html`
  Override this message in all subclasses to render the body-part of the template. Do

all the rendering within aBlock passed to the message `#renderDivFor:on:with:` to ensure that the XHTML environment is properly set-up and that the design can be specified using css-stylesheets.

- `TemplateBody>>renderConfigWith: anAction on: html`
  If you override this message in your subclasses don't forget to call super, as there are the default properties for the title and the css-id rendered in here.

In the appendix you can find the source of the default style-sheet used to layout the components of SmallWiki. Take it as a starting point to create your own design.

3.6 Storage

The abstract storage class provides a protocol to all kinds of storage mechanism implementing persistence in a wiki. It takes care of the notification of changes. Subclasses should implement one of the message `#changed` or `#changed:` to make the given structure persistent.

notification

- `Storage>>changed`
  This message is called whenever something changed inside the wiki structure, if you need to know what exactly happened override `#changed:` instead.

- `Storage>>changed: aStructure`
  Everytime a structure inside the wiki-tree changes this message is called. Override it to provide a storage mechanism.

3.6.1 Snapshot Storage

The class `SnapshotStorage` provides an interface to make snapshots of wikis on a regular bases. With the implementation of the `ImageStorage` as a concrete implementation this is the most secure and most widely used storage mechanism.

accessing

- `SnapshotStorage>>delay: aDelayInSeconds`
  Set the delay in seconds between the snapshots.

- `SnapshotStorage>>lastchange`
  Return the timestamp of the last change of the whole wiki.

- `SnapshotStorage>>lastsnapshot`
  Return the timestamp when the last successfull snapshot has been made.

snapshot

- `SnapshotStorage>>privatePostSnapshot`
  Override this message with code that should be executed after doing the actual snapshot.
- `SnapshotStorage>>privatePreSnapshot`
  Override this message with code that should be executed before doing the actual snapshot.
- `SnapshotStorage>>privateSnapshot`
  Override this message to do the actual snapshot.
- `SnapshotStorage>>snapshot`
  Do not override this message, that simply calls the messages `#privatePreSnapshot`, `#privateSnapshot` and `#privatePostSnapshot` in order to save the wiki structure as a whole.

testing

- `SnapshotStorage>>isChanged`
  Return true if the wiki has been changed since the last snapshot.
- `SnapshotStorage>>isExpired`
  Return true if the delay has been expired since the last snapshot.
- `SnapshotStorage>>isSnapshotNeeded`
  Return true if a snapshot is needed.
- `SnapshotStorage>>isThreadRunning`
  Return if the storage thread is running.

3.7 HTML and Callbacks

Creating valid XHTML is an error prone task when using string concatenation. SmallWiki follows the design of Seaside [4] and implements the class `HtmlWriteStream`. This class subclasses `WriteStream` and it provides a lot of additional messages to append text and XHTML elements to the document being rendered.

The following example could be part of the message `#renderContent` within the `Action` hierarchy to render a simple user-interface:

```
html heading: 'Title' level: 1.
html paragraph: [
    html text: 'Click '.
    html
        anchor: 'here'
        to: self url
        callback: [ :action | action doSomething ] ]
```

The code produces something like the following output:

```
<h1>Title<h1>
<p>
  Click <a href="/?action=ActionClass&callback=31415">here</a>
</p>
```

The first message `#heading:level:` produces a simple section heading of level 1. The message `#paragraph:` is similar to the one of the heading, but in this example we do not pass a string but block: everything done within that block will be put inside the paragraph tags. This mechanism assures that all tags are closed properly and that always valid XHTML is generated. The message `#text:` does basically the same as `#nextPutAll:`, additionally it escapes the given string to make sure the code can be displayed correctly within the web-browser.

The message `#anchor:to:callback:` is probably the most interesting one as it is used to generate an anchor with an assigned call-back block. The `anchor:` argument obviously renders the things that should be rendered as the content of the link. The `to:` argument specifies the place where the callback should be handled: usually this is within the same action, but occasionally you might need to specify something else. The `callback:` argument is evaluated when clicking the link. As an argument the block receives the action that is executing the callbacks, note that this is not necessarily the same action that rendered the link and that is referenced using the keyword `self`.

`HtmlWriteStream` doesn't emit any unnecessary spaces into the output stream, what makes investigation in the HTML code somehow difficult. For this purpose, there is the possibility to enable the included pretty-printer, but keep in mind that this slows down the rendering process and might have unwanted effects on the output in the web-browser.

```
HtmlWriteStream prettyPrint: true
```

To see a more advanced examples about html-rendering and callbacks, have a look at the action-class `CallbackDemo`, that is part of the examples-bundle. Point your web-browser to `http://localhost:8080/?action=CallbackDemo`, play around with the user-interface and have a look at the implementation.

3.8 Testing

One of the main benefits of SmallWiki is that it is extensively tested. Before reporting any bugs you should always run the tests in order to verify if there is something wrong with your set-up. Maintenance, porting or extending should go together with running the existing tests and writing new ones to further improve the quality of the code. All the major releases of SmallWiki have to pass all of the provided test-cases.

The following table is built automatically, while putting together the SmallWiki documentation. This information is taken from the authors current development environment and therefore should be always accurate:

SmallWiki:	`SmallWiki 1.0`
VisualWorks:	`VisualWorks, Release 7.1 of March 21, 2003`
Time stamp:	`November 12, 2003 10:00:20.433`
Test results:	`217 run, 217 passed, 0 failed, 0 errors`

Chapter 4

Extending SmallWiki

This chapter gives a couple of examples about extending SmallWiki. All the given examples are included within the bundle *SmallWiki Examples*, so make sure to unload that one before proceeding. After having written your own code you might want have a look at those implementations, to see some more advanced features not mentioned in this tutorial.

4.1 Statistical Component

The task is to create a template-component listing the type and number of children of the current structure and all its children. To achieve this behaviour we need to create a new template-component that will be included in the wiki and a visitor collecting the information from the wiki-tree.

First of all, let's create a new package called *SmallWiki Example Statistic* to keep all our source code together. We will start out coding the Visitor as this component might already be tested before finishing the template component. Inside the namespace `SmallWiki` create a new class called `VisitorStatistics` that is a subclass of `Visitor`. Add the instance variable `children`, this is the place where we want to store the collected statistical data:

```
Smalltalk.SmallWiki defineClass: #VisitorStatistics
       superclass: #{SmallWiki.Visitor}
       indexedType: #none
       private: false
       instanceVariableNames: 'children '
       classInstanceVariableNames: ''
       imports: ''
       category: 'SmallWiki Example Statistic'
```

Figure 4.1: Statistical Component

Next we need an initializer and an accessor for the instance-variable we just created. Add the following two messages:

```
VisitorStatistics>>children
	^children

VisitorStatistics>>initialize
	children := Dictionary new
```

The next message we need to implement is `#acceptStructure:` that is called whenever the visitor passes a structure and where we want to collect the statistical data. When implementing this message don't forget to call super, to make sure that also all the children of the structure are visited:

```
VisitorStatistics>>acceptStructure: aStructure
	| count |
	count := children at: aStructure class ifAbsent: [ 0 ].
```

```
        children at: aStructure class put: count + 1.
        super acceptStructure: aStructure
```

That's it for the visitor part: let's try out the code in the workspace where you started the SmallWiki server. Inspect the following expression and you should get a dictionary containing all the structure-classes as keys and the number of occurrences in your wiki as value.

```
VisitorStatistics new
        visit: server root;
        children
```

Now, let's create a new subclass of `TemplateBody` called `TemplateBodyStatistic`

```
Smalltalk.SmallWiki defineClass: #TemplateBodyStatistic
        superclass: #{SmallWiki.TemplateBody}
        indexedType: #none
        private: false
        instanceVariableNames: ''
        classInstanceVariableNames: ''
        imports: ''
        category: 'SmallWiki Example Statistic'
```

and add the following message, that is reusing the code we just typed in the workspace and is performing the rendering:

```
TemplateBodyStatistic>>renderBodyWith: anAction on: html
        | result |
        result := VisitorStatistics new
            visit: anAction structure;
            children.
        self renderDivFor: anAction on: html with: [
            html table: [
                result keysAndValuesDo: [ :key :value |
                    html tableRowWith: key title with: value ] ] ]
```

We are almost done! Unfortunately we are not able to add the template within the editor in the web-browser yet, as we forgot to give our template a proper name. Implement the following message on the class side of your template:

```
TemplateBodyStatistic class>>title
        ^'Statistics'
```

Go back to the web-browser, open the template-editor and add the newly created component to the root of your wiki. It should immediately appear at the bottom of the page and give the same result as you have seen in the inspector just a few minutes earlier. If you go down the sub-folders of your wiki, you will see that the numbers change accordingly, as they are dynamically recalculated every-time the page is rendered.

4.2 Information Action

Figure 4.2: Information Action

This tutorial demonstrates how to create a custom action displaying some general information in the web-browser. First of all let's create a new bundle called *SmallWiki Example Info* where we put the following class skeleton:

```
Smalltalk.SmallWiki defineClass: #InfoAction
       superclass: #{SmallWiki.Action}
       indexedType: #none
       private: false
       instanceVariableNames: ''
       classInstanceVariableNames: ''
       imports: ''
       category: 'SmallWiki Example Info'
```

Similar to a template, each action needs to have a title. Action titles are used at several places, to let the user identify the purpose of this class, like in the action-menu or in the

template-editor. Add the following message to the class-side of your code:

```
InfoAction class>>title
       ^'Info'
```

Your action is already useable, but you have to know the exact URL to call it. Point your web-browser to the following URL and have a look at the empty page that will be generated. Note that the tile specified is used to build the heading of the page.

```
http://localhost:8080/?action=InfoAction
```

To add content to this page we have to override the message `#renderContent`. This is just a simple example rendering a table with some system information, but you might want to display something more interesting. An idea would be to use the visitor written in the previous example and present the collected information in this action.

```
InfoAction>>renderContent
       html table: [
           html
               tableRowWith: 'Operating System'
               with: OSHandle currentPlatformID.
           html
               tableRowWith: 'Platform'
               with: SystemUtils version.
           html
               tableRowWith: 'Server'
               with: VersionString ]
```

When hitting the refresh-button you should see the generated content immediately. Unfortunately there is still no button listed calling this action from within any page of SmallWiki. First of all you should register your action within all the structures that this action might be used with. As our action does not depend on a specific structure add the following code to the class-side of `InfoAction`

```
InfoAction class>>initialize
       Structure withAllSubclasses do: [ :each |
          each registerAction: self ]
```

and evaluate the code. When going to the template-editor, choosing the settings panel and adding your action to the list of selected actions, it will immediately appear on all the rendered pages.

Appendix A

Contributing SmallWiki

Join the SmallWiki community by subscribing to the mailing-list. This is the place where you might read the latest news, announce extensions that you have written, ask questions, and discuss about the development and the daily use of SmallWiki.

```
http://www.iam.unibe.ch/cgi-bin/majordomo
```

The home-page of SmallWiki is the place where you are able to download the latest version of this documentation and all the style-sheets will be published there:

```
http://www.iam.unibe.ch/~scg/smallwiki
```

Appendix B

Standard Style-Sheet

```
/* The MIT License
 *
 * Copyright (c) 2003 Lukas Renggli
 * Copyright (c) 2003 Software Composition Group, University of Berne
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is furnished
 * to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

/* -- basic -- */
html, body, div {
```

```
    margin: 0px;
}

img {
    border: 0px none white;
}

body {
    background: white url(header.png);
    background-repeat: no-repeat;
    color: black;
    font-size: 12px;
    font-family: Verdana, Arial, Helvetica, sans-serif;
}

a, a:link, a:active, a:visited, a:focus, a:hover {
    color: #4164AE;
}

a, a:link, a:visited {
    text-decoration: none;
}

a:active, a:focus, a:hover {
    text-decoration: underline;
}

/* -- container -- */
#container {
    margin-left: 5px;
    margin-right: 5px;
    padding-top: 75px;
}

/* -- title -- */
#title {
    text-align: left;
    position: absolute;
    height: 58px;
    top: 0px;
```

```
    left: 0px;
}

#title h1 {
    color: white;
    font-size: 38px;
    margin-top: 0px;
    margin-left: 5px;
}

/* -- session -- */
#session {
    text-align: right;
    position: absolute;
    height: 58px;
    top: 0px;
    right: 5px;
}

#session h1 {
    display: none;
}

#session h2 {
    font-size: 11px;
    display: inline;
}

#session h2:after {
    content: ":";
}

#session .list {
    display: inline;
}

#session .listItem {
    display: inline;
    margin: 0px 0px 0px 5px;
}
```

```
#session .listSeparator:after {
    content: ",";
}

/* -- path -- */
#path {
    text-align: left;
    font-size: 10px;
    position: absolute;
    top: 58px;
    left: 3px;
}

#path h1 {
    display: none;
}

#path .list {
    margin: 0px;
}

#path .listItem {
    display: inline;
    margin: 0px 2px 0px 2px;
}

#path .listSeparator:after {
    content: "/";
}

/* -- actions -- */
#actions {
    text-align: right;
    font-size: 10px;
    position: absolute;
    top: 58px;
    right: 3px;
}
```

```
#actions h1 {
    display: none;
}

#actions .list {
    margin: 0px;
}

#actions .listItem {
    display: inline;
    margin: 0px 2px 0px 2px;
}

#actions .listSeparator:after {
    content: "|";
}

/* -- validator -- */
#validator {
    text-align: right;
}

#validator h1 {
    display: none;
}

#validator .listItem {
    display: inline;
}

/* -- content -- */
#contents th {
    text-align: left;
}

#contents hr {
    border: 1px solid #4164AE;
}

/* -- error -- */
```

```
#error {
    color: #AE4156;
}
```

Bibliography

[1] Sherman R. Alpert, Kyle Brown, and Bobby Woolf. *The Design Patterns Smalltalk Companion*. Addison Wesley, 1998.

[2] Kent Beck. *Smalltalk Best Practice Patterns*. Prentice-Hall, 1997.

[3] John Brant and Don Roberts. Smalltalk Compiler-Compiler (SmaCC). http://www.refactory.com/Software/SmaCC.

[4] Avi Bryant and Julian Fitzell. Seaside. http://www.beta4.com/seaside2.

[5] Randy Kramer. Wiki Engines. http://c2.com/cgi/wiki?WikiEngines.

[6] Camp Smalltalk Project. Smalltalk Web Application Zoo (Swazoo). http://swazoo.sourceforge.net.

[7] Masashi Umezawa. Smalltalk Instance eXchange in XML (SIXX). http://www.mars.dti.ne.jp/∼umejava/smalltalk/sixx.

[8] Cincom VisualWorks Smalltalk. http://www.cincom.com/scripts/smalltalk.dll/.