Project Cadmus

SWEEP System Requirements

and Proposed High-Level Design

Version 0.2

Marathon Beach, Saturday, 15 November 2003

By Fotios Basagiannis

For Miltiades and the Athenians

Many thanks to Luke Barlow and the NKD crew

Preamble and Disclaimer

This document is only an initial and tentative technical specification of the SWEEP system. This document should mature over a considerable period of time with input from various people before any serious formal design effort can begin.

As long as the version of this document is below 1.0 it is not to be considered complete.

I am sure that a lot of particulars can be changed or improved in this initial specification. However I am convinced that, keeping the basic tenets of the SWEEP system, one can build a very capable, stable, extensible and cross-browser system with few problems.

Motivation

It is highly desirable for web authors (a term that has expanded to enfold, to some extend, almost every web user) to be able to easily create or update web content from any web connected and enabled host and without the need of specialized desktop based software.

The usual way of accomplishing this is via web applications with various degrees of capability and complexity. The delivered application usually enables users to create and edit web content in remote file systems using just a web browser capable of running Java applets, ActiveX applets or even just JavaScript.

Of the three aforementioned client-side programming technologies it is JavaScript that is the most desirable and yet the more technically challenging to use in the particular domain of web based content management systems.

JavaScript is mainly desirable because it does not require the user to have software installation permissions on the box he is using to connect to the Internet. Further, JavaScript (as a script language) usually requires less development time than languages like C++ to deliver the same end product.

JavaScript, however, has some important disadvantages:

- It cannot do as much as languages like C++ nor does it have deep API access.
- It is relatively slow in its execution.
- JavaScript and especially DOM APIs have significant differences in both syntax and semantics among existing browsers.

The implied challenge to the CMS software engineer is to design and implement a JavaScript based Web CMS system that overcomes JavaScript's inherent inefficiencies while providing powerful web based content management.

SWEEP Introduction

SWEEP is an acronym deriving from "System for Web Entity Editing and Publishing".

SWEEP is based on the seminal concept that the DOM API should not be used or relied upon for GUI based HTML editing. Instead, elements should be manipulated via appropriate string manipulation and reinterpretation of their HTML code.

SWEEP relies in the fact that all major browsers support the innerHTML property of elements. It also relies in the fact that all major browsers support hooking of events.

What SWEEP does is hook onto a page's elements of interest and then - based on an extensible and fully customizable set of XML definitions of element manipulation operations and related GUI based user interactions - manipulate the element's HTML. The element is then reinitialized and displays in its new state.

These XML definitions are called NOMADs - (NO)de (MA)nipulation (D)efinitions. NOMADs specify GUI elements and their semantics of their use in relation to element manipulation. NOMAD interpretation is handled by SWEEP's NOMAD interpreter.

Permissions can also be applied that determine what a user can and cannot do to an element or if he/she can do anything at all. Each file at the disposal of the SWEEP system is identified by its unique URI and is associated with a page profile object (in the OODBMS) that associates element ids, users and permissions. All HTML elements are potentially editable in a manageable and granular manner.

SWEEP aspires to become the best-ever Web based CMS with a client side that uses only cross-browser JavaScript

Web Entities

SWEEP treats the web as a universe of distinct web entities that organize themselves into interlinked web pages. A web entity is basically what corresponds to an HTML tag or what DOM specs call a document element. It naturally follows that we may have web entities that are totally unrelated and we may have entities that are laid out in a nested fashion.

Each entity is identified by a universally unique string that is basically the URL to the web page the web entity belongs to plus the entity's XPath as the URL's query string.

> e.g. http://www.mydomain.com/mypage.html?/body[1] identifies the first child element of this particular page's body.
>
> Note: If the URI query string is required to use only non-reserved URI characters, it can be URI encoded.

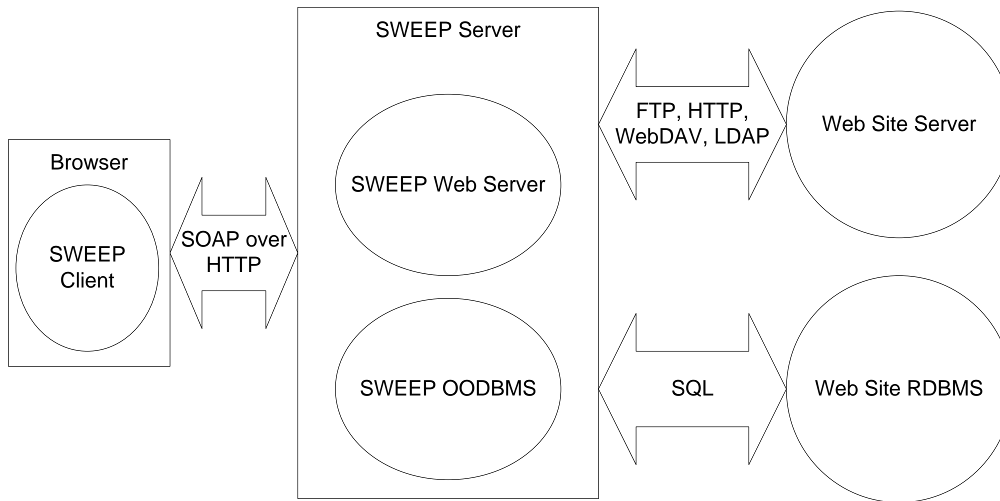
By default, SWEEP treats this universe of web entities as a permissions sensitive hierarchical structure. The SWEEP permissions model is modelled after the NTFS file system (to a significant degree).

Basic SWEEP architecture

Sweep is a 3-tier system that uses SOAP over HTTP for client-server communications and is capable of interfacing with remote file systems and databases.

SWEEP consists of:

- A SWEEP server, which comprises proprietary web server and OODBMS system (probably built in C++).
- A SWEEP client fully built using cross-browser JavaScript.



The server will expose a public interface comprising methods from various server side objects. The client will be able to make remote calls to these methods using SOAP over HTTP.

Each web entity is appropriately represented by a persistent object on the server side and is associated with a permissions object that points to various user objects thus recording permissions.

.

What makes SWEEP special?

- Minimal browser requirements (e.g. no built in element editing capabilities whatsoever are required by the browser)
- Very significant DOM API agnosticism (only event hooking and DOM hierarchy traversal is done, and that in a cross-browser manner)
- Cross browser operation
- Extensibility/Modification of the system's web entity editing capabilities by the use of a declarative markup language and maybe the use of simple embedded scripts. (NOMADs)
- Allows editing of pages that make heavy use of JavaScript with no problems.
- Well abstracted and structured software architecture (Kernel, Components, APIs, well specified development framework for NOMAD developers)
- Strong browser-independent support of major W3C standards (HTML, XHTML, XML, XPath, SOAP) as well as regular expressions.
- Robust HTML source code style preservation and customization (SWEEP does not mess up your HTML).
- Generated HTML code complies with either user customizable style guides or W3C standards like XHTML
- Fully proprietary web server and OODBMS that are built to fit the needs of the system and can be customized per case for maximum efficiency, while safeguarding the investment of the developers by giving them full control of the platform on which they develop.
- Use of original and ground-breaking JavaScript based algorithms and frameworks for providing functionality that was not previously available by JavaScript based systems (e.g. the Mjollnir technology)
- Secure user authentication using SWEEP's own security system: S/Key Commando

Some SWEEP User Requirements

- Ability to edit static html pages (whether they contain javascript or not) and then save them back to their remote file systems.
- Provide listing of remote files
- Ability to graphically create xml hierarchies (for site menu building or other purposes)
- Ability to define and view db schemas on the server side
- Support HTML templates
- SWEEP should ideally be able to create and maintain any kind of web page
    - static html
    - client-side dynamic (js and dom)
    - server-side dynamic (asp, jsp, php)
- Can do both text and GUI based HTML editing
- GUI is customizable and xml specifiable
- Displays URI of current page (URI is the id of the page) and XPath to current element
- Displays permissions for each selected element
- Selected element and editable elements are graphically marked (border?)
- Has editing and preview modes (Preview removes SWEEP borders and enables javascript)
- The interface will permit hierarchical element browsing:
    - Prev-next sibling node
    - Parent-child node
- Element creation is also supported. You can select from a list of element types. The new elements are created where the pseudo-caret is located or right after the presently selected element.
- Some elements are auto-created as you type (e.g. another <LI> when you type <enter> right after the creation of the first LI)
- At any time, the user should be able to display the source code of any element or that of the whole page (with syntax highlighting).
- Certain users login as administrators and have access to the admin interface that gives them permission setting facilities.
- There will be a history of selected nodes that you can travel back and forth (as well as the standard DOM based element navigation)
- SWEEP will have a file listing and will support page and resource drag-drop operations.
- SWEEP may also be used for XML editing with well-formedness and validity checks (xmlSpy style).
- Extensible syntax-highlighting facilities (SHDs – Syntax Highlighting Definitions)
- Capability to edit XHTML in a way that is compliant to user supplied XML schemas - XHTML applications

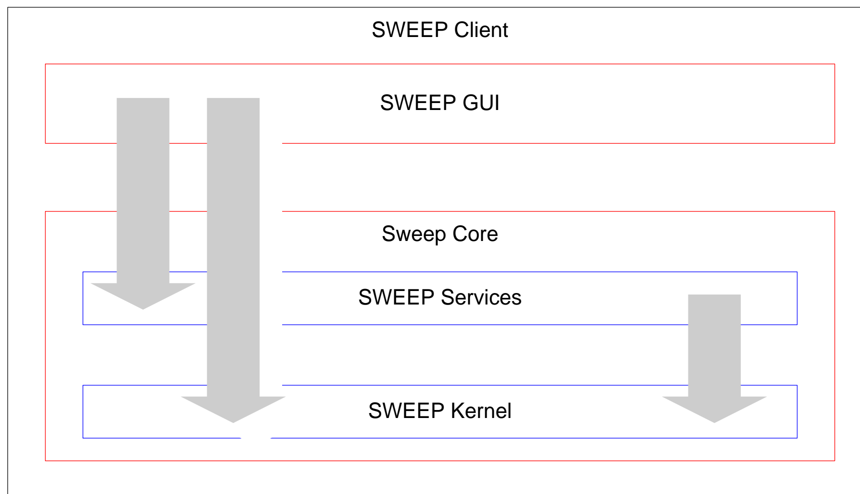
Some SWEEP Software Design Requirements

- All subcomponents should expose a public API.
- Abstract file system interface (that interfaces to remote servers using a variety of protocols)

The SWEEP Client

The SWEEP Client comprises

- a layered core of services
- extensible/customizable GUI system



GUI commands are processed by NOMADs and appropriate SWEEP core calls are made.

The document source is kept separately and updated appropriately every time an element's innerHTML is altered and reinitialized.

Note: SWEEP core layers could also be implemented, for MSIE only, as an ActiveX - for extra speed or code obfuscation purposes during product demonstrations or even SWEEP use in Intranets. Since SWEEP relies more on string processing than on DOM manipulation, its implementation in C++/ATL would be relatively easy.

The minimum requirements that the SWEEP client will have from a browser will be:

- read/write support of the innerHTML property of elements
- JavaScript support
- DOM support for event hooking
- Basic DOM support for simple sibling-parent-child element navigation (using the arrow keys or arrow buttons)

SWEEP Client Core

The SWEEP client core is a set of objects that expose abstract interfaces to core services that the client relies on for the editing and management of online content.

Client Kernel Layer

- Loading engine – manages/serializes HTTP calls in a queue
- Regular expressions engine – browser independent implementation

Client Services Layer

- innerHTML string processing engine
- S/Key Commando
- XML parser
- NOMAD interpreter
- Mjollnir fragment selection engine
- XPath interpreter
- GUI generator
- Provides a public heap memory (an associative array)
- Stores original document source as a string.

SWEEP Client GUI

The client GUI is basically a page that features

- a server-side file listing
- an IFRAME that is used as a page editing sandbox
- a NOMAD interface display box

Clicking on files in the file listing loads them in the IFRAME for editing.

There may also be customizable toolbars and panels relating to new element creation.

How the SWEEP client works

Check out a very basic SWEEP proof of concept page: http://fotios.cc/projects/sweep/

HTML pre-processing

Pre-processing (and post-processing if needed) of html source will probably be done using the SWEEP regular expressions engine.

The pre-processors main task will be to separate HTML from Script.

Page bootstrapping

- On load, all page scripts and links are disabled - they can be seen at work in preview mode (script/html separation).
- Permission expando properties are attached to elements that carry permissions
- Page elements of interest are hooked

SWEEP may then use its own script logic on various page elements (for user actions like element resizing, element dragging, etc.)

The element events that are usually hooked are:

- onclick
- onmouseover
- onmouseout

These are used by SWEEP in order to provide visual queues about which elements can be edited and in order to allow the user to select the element he wishes to edit.

Event hooking

Event hooking is done in MSIE using the attachEvent method and in mozilla using the addEventListener method. Hooking is done on elements (entities) that the present user is allowed to edit.

innerHTML editing

SWEEP never tries to edit HTML using the DOM as this would mean

- Endless code amendments in order to deal with different browser intricacies in the implementation of the DOM and even the intricacies of DOM manipulation itself.
- A significant degree of loss of the document's original HTML formatting and coding style.

Instead, SWEEP makes all HTML editing by efficiently editing (using the engine in the SWEEP core) the innerHTML of edited elements on the fly (the outerHTML of the element is actually edited but SWEEP gets to it via the innerHTML of the parent, as the outerHTML element property is not available in many browsers).

The original source of the document is fully preserved, including line breaks. The document source remains totally unchanged after a SWEEP editing session (except of course the parts that were edited). This is not the case with many CMS systems (even desktop based ones).

Since, in the live version of an entity's HTML source (as it can be found in the SWEEP sandbox) there may be SWEEP embedded artefacts, the innerHTML that is edited by the NOMAD is loaded from the separately kept pristine document source and after editing:

- It is put back - as it is, in its new state- into the document source
- It receives additional SWEEP artefacts (like permissions properties) and is re-evaluated as the new HTML source of the element in the SWEEP sandbox.

New HTML code is generated according to user defined (and stored as XML) coding style conventions (e.g. whether to use single, double or no quotes for tag attribute values)

BaseURL

Since the page HTML source will be coming from the SWEEP server (or will be initialized under the localhost domain) instead of from its normal server, the baseURL document property will be used in order to make document resources (either domain secured or using relative src urls) load in the SWEEP document's context.

The baseURL change should not be reflected in the original HTML source string - just in the sandbox's HTML.

Adding text to elements

Typing within an element that is a textNode capability should also be accommodated by SWEEP.

This will also be done using innerHTML editing as we do not want to require that the browser we use offers built in element editing capabilities (we want to have SWEEP cover as much browser realty as possible).

Since elements that require text editing capability are quite common the innerHTML processing required for typing text will be supported by SWEEP's core innerHTML processing engine.

Note that using this kind of type-editing we retain complete control over HTML code generation.

The SWEEP Server

-- Server components diagram here

The Sweep Server will eventually use 3-way merging to support concurrent editing of documents.

In initial SWEEP versions, documents will appear locked to users other than the first to open them (concurrent editing will not initially be supported)

The CVS subsystem of the server should ideally support graphical conflict resolution

The OODBMS should probably be coded in C and not Java for efficiency reasons.

Initially (before development of SWEEP's proprietary OODBMS) it can be based on the ObjectStore system or some free OODBMS (there are a couple around).

SWEEP Concepts and Components

All components expose public APIs.

The innerHTML string processing engine

The innerHTML engine guarantees that for every innerHTML manipulation the original page source is appropriately updated (provides source synchronization guarantee).

The innerHTML processing engine should expose a very generic API that consists of adequately abstracted methods.

Methods like setAttribute() and properties like innerText should be supported (in general SWEEP should use a subset of MSIE's DOM methods - the most generic of those methods and properties).

There should also be abstracted inline style manipulation methods.

About Mjollnir (Mee-all-Neer)

Proprietary Mjollnir technology is definitely needed for the extraction of raw html fragments from mouse selections within element.

* Add my mjollnir post from soq list (the one that talks about binary searching)
    - Make mjollnir cross-browser

NOMADs

NOMADs are expressed in a declarative XML based language that is interpreted by an engine that:

- displays the appropriate interaction interface for an HTML tag as specified in the NOMAD
- performs the necessary innerHTML tag code alterations as specified in the NOMAD.

A NOMAD may contain code pertaining to the operation of the particular GUI they present (e.g. mouseover code for their buttons)

A NOMAD does not need to contain code for the manipulation of the element (this is within the optimized engine in the SWEEP core). The NOMAD only needs specify **what** needs to be done when say a button is pressed – not **how** it is to be done.

NOMADs that want to allow for direct user-element interactions (e.g. dragging an element to resize it) will have to declare that as "new_action" and

- declare the element's events that they want to process
- provide script for the handler functions of those events

A NOMAD can specify both GUI and Keyboard interactions between the user and the element being edited.

A NOMAD's GUI will be basically buttons and textfields that help the user specify some desired change on the element being edited. The NOMAD will contain declarative markup code that declares the GUI elements (out of an extensible collection of SWEEP widgets with customizable styles) as well as logic that provides the plumbing between widget events and SWEEP core methods (thus implementing the user-GUI interaction generated imperatives by making the required calls to the SWEEP core engine's abstracted innerHTML string manipulation API that may obey XHTML well-formedness rules or other user preferences based coding style conventions)

NOMAD rendering may be done via:

- XSL processing (implies another requirement for supported browsers)
- Proprietary xml processing. Either:
    - Use browser's built in XML parsing capabilities (implies another requirement for supported browsers)
    - Do the parsing using a proprietary XML engine (implemented in JS) - Recommended (no extra browser requirements)

There is a 1:1 correspondence between HTML tags and NOMADs (with each NOMAD potentially coded separately). This way you divide and conquer the fairly big semantics space of the HTML DTD.

All of SWEEP's NOMADs can be stored in a single xml file with multiple <nomad> tags or in separate files (one per tag)

Each NOMAD may have script code embedded as CDATA sections.

There may be extra provisions like a well-defined declarative sublanguage that will allow the NOMAD scripter to specify the interaction between GUI and Sweep Kernel in a concise and completely code-free way.

A NOMAD will be able to manipulate both the attributes of the whole element as well as text selections using the Mjollnir system.

-- NOMAD sample here

Permissions

Element Permissions are hierarchical and modelled after NTFS permissions. If you have permissions on a parent you have the same permissions on its children nodes by default (these permissions may be overridden however by the child element's own permissions)

The permissions could be internally represented by a bitmask.

-- add my permissions post from Soq list

PUDs

The SWEEP system features an extensible page update system (PUDs) that supports by default FTP, HTTP PUT/DELETE (and later on webDAV and LDAP)

A document id (URI) is associated with a PUD (Page Update definition) that may specify alternative ways of page update (FTP, HTTP, etc.) and should contain access/authentication details for each.

The future of SWEEP

SWEEP will initially be specified and implemented as an editor/manager of static web pages only.

However, it may be viewed as highly desirable that dynamically created pages, somehow, become editable by applying the same metaphors and using the same GUI, that SWEEP uses for static pages; that is, employ ground-breaking HTML reverse engineering technology.

Such a system could be devised by having user GUI based interactions be translated to:

- Appropriate DB actions for page content
- Appropriate code amendments (in HTML or ASP, JSP, PHP, etc.) for page presentation elements.

If we require that initial (manually written) server-side code is largely preserved (changing only the parts of it that are necessary), the latter may only be feasible as long as there is some well standardised way in which dynamic pages are scripted by the web programmers - maybe dynamic pages could be based on HTML templates with embedded calls to functions (coded in a particular server-side script language) that accept all relevant presentation related information as arguments in a particular order.

If we do not require full code preservation, then the server-side code of the page could be rebuilt by using a smart reverse engineering algorithm and having the user supply the particulars of how the back end connection between script and DB is to be accomplished.

In fact, using reverse engineering, SWEEP could be used for building dynamic pages (out of standard HTML building blocks) from scratch, without having to write a single line of code; just drag and drop dynamic HTML components and provide back end connection details.

The former could be accomplished by providing a special version of the page that contains embedded db information for each data area on the page (and is put together by either a web developer or a smart server-side SWEEP utility).

In the case of reverse engineering, the former would be easier to accomplish as the connection and DB details for each data fed page element would be saved in SWEEP's OODBMS since they were first created (or recreated).

Note that the data editing capability requires SWEEP's back end to be able to interface directly to various DBMSs.