Survey of software architecture

Name of system: SocialCalc

Reviewer: Trevor Bekolay

Date: November 21, 2011

Author of software: The original implementation (called WikiCalc) was created by Dan Bricklin. SocialCalc is a rewrite of WikiCalc, written by a team of programmers at Socialtext, led by Dan Bricklin. Audrey Tang, the author of the chapter, is part of the Socialtext team.

Author of book chapter: Audrey Tang

Five star rating of book chapter: The chapter is well written and organized, and gives a good amount of detail, being relatively comprehensive without being too technical. The "lessons learned" section is better developed than in other chapters, though the lack of a clear goal made the chapter as a whole not as well motivated as the Wesnoth chapter. Overall: ****

Purpose of system: SocialCalc is a spreadsheet program that runs in a browser, allowing the spreadsheet to be accessible over the internet and edited by multiple people simultaneously.

While the program is made freely available through an open source license, it is primarily used as part of a suite of "social" tools created by Socialtext and sold as a socially-oriented intranet web content management system to mid-size and large organizations. SocialCalc has also been used as the collaborative spreadsheet program included with the laptops distributed in the One Laptop Per Child program. It is also used by the Drupal module "sheetnode," which uses SocialCalc to make an editable collaborative spreadsheet possible in the popular content management system Drupal.

Basic metrics

Lines of code: Below is the output from the CLOC tool.

Language	files	blank	comment	code
Javascript	8	4429	3273	16763
Perl	4	922	627	4055
HTML	3	208	0	1742
Bourne Shell	10	57	157	253
CSS	1	3	18	27
SUM:	26	5619	4075	22840

In total, with blank lines and comments, SocialCalc is made up of 32.5 KLOC.

Project start-up: WikiCalc development was started by Dan Bricklin in 2005. It was abandoned once SocialCalc development started in 2006. The first release of SocialCalc occurred in October 2009, three years after the start of development.

Number of major releases: Two; SocialCalc is currently on version 1.1.0.

Number of developers: Socialtext has approximately 40 employees worldwide, but it is unclear how many of those employees were involved in SocialCalc development. Given the size of the codebase, very likely a small proportion of those 40 employees.

Size of user community or number of installations: Socialtext lists 80 companies on its list of customers, each of which would have several hundreds or thousands of employees accessing the intranet. Over a million OLPC XO-1s have been sold, each of which has a copy of SocialCalc installed on it. There are currently approximately 140 installations of Drupal using "sheetnode." In all, I would estimate that there are approximately 1.2 million users with access to SocialCalc, though it is unlikely that as many users actually make use of it.

Major stakeholders: The primary stakeholder is Socialtext, which uses SocialCalc as a part of its suite of enterprise web collaboration tools. The people involved with the One Laptop Per Child program are also stakeholders, as they rely on SocialCalc to provide the ability to create spreadsheets on their laptops. Web administrators that use Drupal and the sheetnode module are also minor stakeholders.

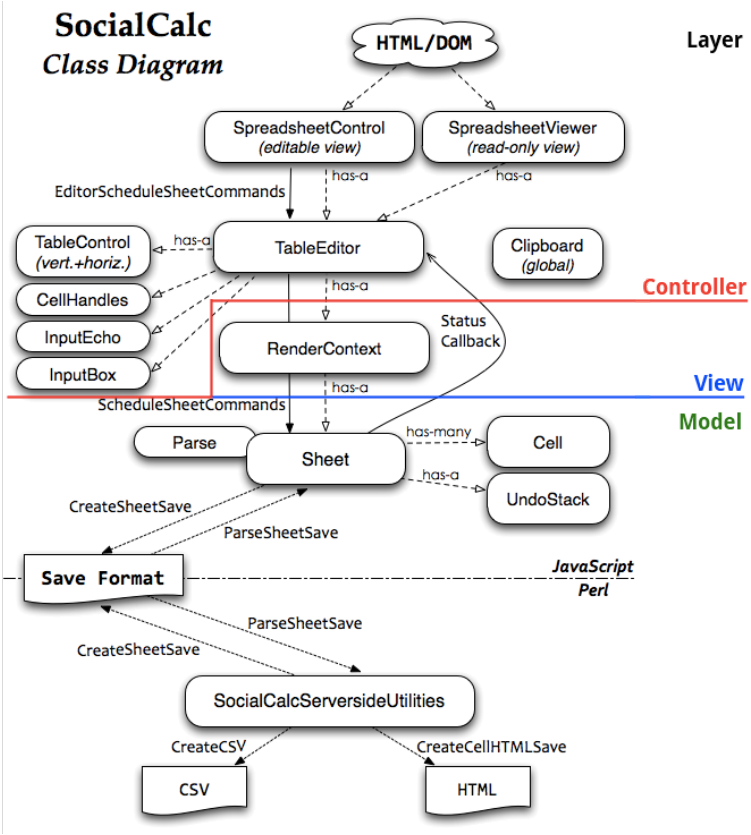
Use of concurrency: SocialCalc is a piece of client-side JavaScript code, with a small server-side component written in Perl. It is not resource intensive enough to require the use of concurrency.

Implementation language: As previously mentioned, the majority of SocialCalc is implemented in JavaScript, and is executed on the client's machine. A smaller portion of code is executed on the server side, and is implemented in Perl, although a port of SocialCalc using Node.js as the client-side language exists.

Supporting software: SocialCalc uses the Javascript library `Wikiwyg` for rendering WikiText. It also uses the `Web::Hippie` for real-time collaboration with multiple users.

High level architecture

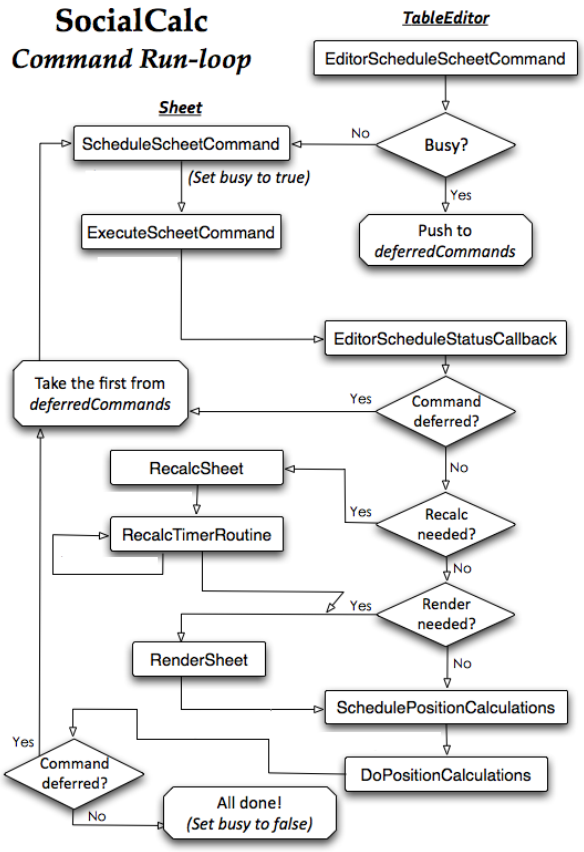
Diagram of software architecture: Because SocialCalc is such a small program, the high-level architecture can be expressed in terms of the JavaScript and Perl classes that make up the program, making it more of a concrete architecutre than a conceptual one. There is no inheritance in SocialCalc, so all of the dotted lines represent ownership relationships; for example, the `TableEditor` object contains a `RenderContext` object.



This diagram comes from the book chapter. I have also added lines and text to indicate which of the classes belong to each of the layers, as the main architectural style used in SocialCalc is layering, with the layers following the Model-View-Controller design pattern. All of the classes in the Model layer deal with the actual storage of the information in a spreadsheet. The class in the View layer deals with what portion of the spreadsheet can be viewed in the UI currently. The classes in the controller deal with handling user interaction with the UI, which can change the View layer (updating the UI to show different portions of the spreadsheet) or the Model layer (updating the actual values stored in the spreadsheet).

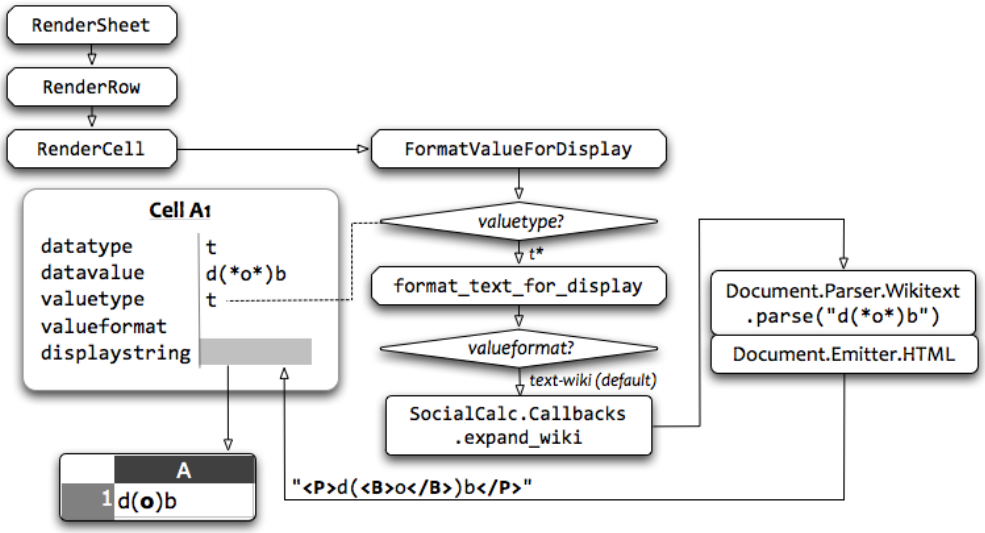
High level scenarios: The chapter describes two important high-level scenarios.

The first is the general scenario of what happens when the user initiates any command through the `TableEditor`. SocialCalc is designed to be responsive, even if the underlying processes are busy, so when a command needs to occur, rather than stopping until the command is run, SocialCalc adds the command to a list of commands to be run in the background, allowing the user to continue making changes to the spreadsheet while the changes are made in the background. The flowchart for doing this is included below (simplified from the book chapter).



The important element here is the *deferredCommands* queue. If the engine is currently busy, the current command is added to the *deferredCommands* queue. Once a command is completed, the *deferredCommands* queue is checked; if an element exists in the queue, then it is executed before the engine is marked as not busy.

The second scenario is how a cell is rendered, specifically how a cell containing text formatted with Wiki-markup is rendered. The ability to use Wiki-markup is another feature that sets SocialCalc apart from traditional spreadsheet programs, as well as other online spreadsheet appications.



The intermediate step `SocialCalc.Callbacks.expand_wiki` is not strictly necessary, and the parsing step could happen immediately after the valueformat is determined to be text-wiki. However, using this Callback mechanism makes SocialCalc more modular, as it is possible to use a different mechanism to implement this callback, and can easily be made to do nothing if some users of SocialCalc do not want to enable WikiText in their SocialCalc installation.

Data structures or algorithms: The save format used by SocialCalc is quite different from a typical spreadsheet program. Rather than just being one large file, the save format is split into four parts, two of which are optional. This save format has the advantage of being human readable, while still being easy to manipulate programmatically. The four parts that make up the save format are as follows.

- The required `meta` part lists which of the other parts are included.
- The required `sheet` part contains the information about the format and content of each cell in the spreadsheet. It also includes some information about the entire spreadsheet, such as the fonts used, and other formatting information.
- The optional `edit` part saves the `TableEditor`'s state, so that the spreadsheet looks exactly the same when loaded.
- The optional `audit` part contains a partial history of commands executed.

As mentioned in the previous section, the *deferredCommands* queue is an important data structure that allows SocialCalc to remain responsive even when a number of commands need to be executed by the engine. Another related data structure that is interesting is the undo stack, which is only briefly mentioned in the chapter. Whenever a command is executed, or added to the *deferredCommands* queue, a corresponding undo-command is added to the undo stack. When the user clicks Undo, the undo-command at the top of the stack is run to restore the application to the state before the last command was executed.

The undo stack is also used in a clever manner for conflict resolution when multiple users are editing the same spreadsheet. This will be discussed in the "real-time" section below.

Architectural style: The primary arichtectural style is layering. The particular layers used fit a common design pattern, the Model-View-Controller pattern. The mapping between layers and classes is straightforward: `Sheet` is the model, representing the data in the spreadsheet in memory; `RenderContext` is the view, keeping track of which parts of the spreadsheet should be displayed on the page; and `TableControl` is the controller, taking in mouse and keyboard input, updating the view and model when necessary.

Major evolutionary changes: The biggest chance occurred in 2006 when Dan Bricklin decided to work with Socialtext, abandoning the feature-complete WikiCalc for a complete rewrite called SocialCalc. The primary architectural change between the two was in the roles of the client and server. In WikiCalc, all of the processing was done server-side by several perl scripts. This meant that everything had to be processed behind the scenes and rendered all at once, which meant poor performance and poor interaction. SocialCalc moved the majority of the processing to the client-side, meaning that the server only had to be used sparingly, making the whole application more responsive.

Performance bottlenecks: In the original WikiCalc, a large bottleneck was in the rendering of large spreadsheets, as each cell in the spreadsheet was a table cell entry in the browser's document object model (DOM); for even moderate sized spreadsheets (say 100 rows and 100 columns) that meant many DOM entries (10,000 for a 100×100 sheet). SocialCalc removed the bottleneck by having a predefined numbers of columns and rows displayed, and when the user scrolls to different parts of the spreadsheet, the displayed rows and columns are updated with information being scrolled to. Additional DOM entries are not added. This means that the application is always only representing a subset of the whole spreadsheet in the DOM, even though the whole spreadsheet is available through the `Sheet` class.

Real time: Real-time collaboration is an important part of SocialCalc. When multiple users are editing the same spreadsheet, SocialCalc adds an additional step to command-running scenario described above; after the command is removed from the *deferredCommands* queue, but before the command is actually executed, it is broadcast to any other users editing the same spreadsheet.

This handles the vast majority of online interactions, but with this strategy there is the possibility that two users will change the same cell of the spreadsheet at the same time, causing a race condition in which users could end up with different versions of the same spreadsheet. To handle this conflict, the built-in undo/redo mechanism is used. As a client broadcasts its command, it checks to see if there is a command on the same cell currently pending. If there is, then it undoes the current command, and then redoes the command once the pending operation is completed. That means that if two clients make a change at around the same time, whichever one occurs last will always be the command that is reflected in both clients.

Notation for architecture: In general, in the diagrams above, nodes are classes and arrows represent either the flow of control or has-a relationships. The particulars of each diagram should be clear upon inspection.

Methodology: Socialtext uses an agile methodology. This is slightly complicated by the fact that the development team is not located in the same physical area; in Socialtext, team members are located across the United States, and indeed the world (chapter author Audrey Tang is located in Taiwan, for example). Tang notes, however, that rather than being a complication, the "distributed agile" methodology used by Socialtext results in more self-descriptive artifacts (e.g. code and tests), and quicker development because an entire Design-Development-Quality Assurance cycle can happen in a single 24-hour day with team members located in time zones 8-hours apart from each other.

Appendix: Kruchten's context attributes

Size S (32.5 KLOC)

Criticality Med (an important spreadsheet might need to be viewed at a critical time)

Age of system M

Rate of change Lo

Business model Commercial (even though the software is released with an open source license)

Stable architecture Lo

Team distribution VHi (different time zones)

Governance Med