Case Study of SocialCalc

Shaohua Shen

The first spreadsheet program, VisiCalc, was conceived by Dan Bricklin in 1978 and shipped in 1979. This earliest one was created with quite straightforward concept which was a table spans infinitely in two dimensions with cells populated with various elements. However, it managed to have many applications including accounting, inventory, list management and many other ones. But collaboration was particularly hard under the file-based model. To address these issues, wiki model was invented by Ward Cunningham in 1994 and inspired Dan Bricklin to start working on WikiCalc in 2005. Then in 2006, he teamed up with Socialtext and began developing SocialCalc in Javascript through rewriting WikiCalc. This rewrite's target were large, distributed collaborations, and sought to deliver a more desktop app style look and feel. Other design goals included: "1. Capable of handling hundreds of thousands of cells; 2. Fast turnaround time for edit operations; 3. Client-side audit trail and undo/redo stack; 4. Better use of Javascript and CSS to provide full-fledged layout functionality; 5. Cross-browser support, despite the more extensive use of responsive Javascript"[1]. In 2009, SocialCalc was released after three years of development and several beta releases. It meted the design goals successfully.

[2][3]SocialCalc's interface and classes can be presented in two figures:

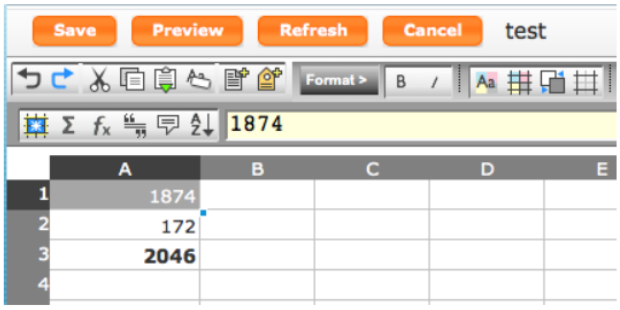

Figure 19.5: SocialCalc Interface

The Javascript components were designed with a layered Model/View/Controller style. As shown in the figure, there are various classes, including *Sheet, Cell, RenderContext, TableControl, SpreadSheetControl and SpreadSheetViewer*.

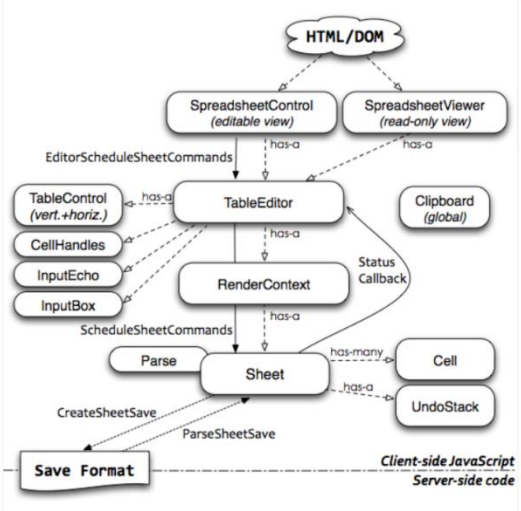

Figure 19.6: SocialCalc Class Diagram

SocialCalc does all recalculation and DOM updates in the background, improving responsiveness, and thus allows the user to keep changing several cells at the same time when the engine catches up on earlier changes in the command queue. All operations are recorded in an audit log since all commands in run-loop are saved as they are executed. The *sheet.CreateAuditString* method offers an audit trail. Besides, an undo command will be created for each command executed.

---

[1] 19.1. WikiCalc, SocialCalc by Audrey Tang

[2] Figure 19.5, SocialCalc by Audrey Tang

[3] Figure 19.6, SocialCalc by Audrey Tang

When talking about Object-Oriented programming design, we cannot avoid discussing SOLID principles. Initially, SocialCalc follows single responsibility principle, with each class focusing on a single aspect: "1. *Sheet* is the data model, representing an in-memory structure of a spreadsheet; 2. *Cell* represents a cell's content and formats; 3. *RenderContext* implements the view, being responsible for rendering a sheet into DOM objects; 4. *TableControl* is the main controller, accepting mouse and keyboard events; 5. *SpreadSheetViewer* is an alternate top-level UI that provides a read-only interactive view; 6. *SpreadSheetControl* is the top-level UI with toolbars, status bars, dialog boxes and color pickers"[4]. SocialCalc also follows Open-closed Principle. By adding names callbacks into the *SocailCalc.SheetCommandInfo.CmdExtensionCallbacks* object, the application embedding SocialCalc are able to define their own extra commands and use the *startcmdextension* command to invoke them. According to the class diagram above and explanation of different classes in the article, it is obvious that SocialCalc meets Liskov substitution principle. *Cell* class is a subclass of *Sheet* class since in SocialCalc, a sheet consists of a number of cells. So, *Cell* class is substitutable for *Sheet* class, the behavior of the set of cells can represent a sheet. And for the same reason, SocialCalc doesn't follow Dependency inversion principle. From the class diagram, we can see that SocialCalc divides multipurpose interfaces, such as controllers, into small focused ones. Thus, it follows Interface segregation principle.

Then discuss another set of principles, GRASP. From the contents in the article, it is apparent that SocialCalc meets information expert principle and creator principle. For example, for the class *Sheet*, its properties allow it to represent a structure of a complete spreadsheet, thus in SocialCalc, the responsibility is assigned to *Sheet* class. Furthermore, cells aggregate sheet, cells closely use sheet and have the initializing data for sheet, thus creating *Sheet* class is assigned to *Cell* class. Besides, coupling remains low in SocialCalc since it assigns responsibilities to many different classes. This meets the requirement of low coupling principle. In SocialCalc, we have *TableControl* class for table editor UI and *SpreadSheetControl* class as the top-level UI. This follows controller principle. Also, objects in SocialCalc are designed to be manageable, focused, understandable and support low coupling. Thus, the cohesion remains high in SocialCalc. This satisfies high cohesion principle.

As mentioned before, SocialCalc does all recalculation and DOM updates in the background, which improve its responsiveness. Additionally, as SocialCalc succeeds in providing rich-text editing, real-time collaboration through commands broadcasting and cross-browser and cross operating systems transport. In addition, SocialClac has built-in undo/redo mechanism for solving editing conflict during collaboration. It also allows defining extra commands and other plugins. With these features, the functionality, usability, reliability, performance and supportability of SocialCalc are above average.

In conclusion, with all the discussion above. SocialCalc follows most of the Object-oriented programming design principles and has nice FURPS. I believe the design is a good one. And meets its original goals.

[4] 19.2. SocialCalc, SocialCalc by Audrey Tang