# A Flexible Architecture for Customizable Web Based Spreadsheet Engine[1]

Ivo Marinchev[2]

*Abstract* – **In this paper we introduce a flexible architecture for development of scalable, and customizable web based spreadsheet engines. Our architecture is geared towards new trends and technologies in application development as web services, rich clients, AJAX, web applications, etc. Finally we present our current implementation developed as AJAX based web application that uses relational database for persistence storage.**

*Keywords* – **spreadsheet engine, AJAX, web applications, web services, rich-clients, web 2.0.**

## I. Introduction

In the last few years there is a trend to "web enabling" (applications that can be started and run through web browser) of the current desktop software applications. New ones are directly built as web application and many old one are re-implemented or extended as web based.

The first distributed applications with centralized business logic (backend) were rich-client applications. They require more sophisticated and responsive user interface (than what HTML, CSS, and JavaScript can offer at that time) that run on the user's computer and communicates with a business logic that is physically located on the centralized servers. Rich client applications and technologies appear about 10 years ago but could not become widespread. They remain in use mainly in intranet environment inside the organizations or shared between affiliate organizations. Their failure to become widespread was due to many factors some of which are:

- they require the users to install additional software on their systems;
- security concerns;
- not having enough support from big application vendors;
- high price tag.

The most prominent technologies in this area are Microsoft's ActiveX, Java Web Start (JWS), Eclipse Rich Client Platform (Eclipse RCP), and Macromedia Flex.

Recently widespread adoption of web applications became feasible with the introduction of several key technologies in practically all of the modern web browsers - Internet Explorer, Firefox, Mozilla, Opera, Safari, and Konqueror. These technologies are CSS [3, 4], JavaScript [5], DOM [6] and DHTML (dynamic screen re-flow). Although most of them had reliable implementations even in year 2000 the biggest boost started just recently with the introduction of so named XMLHttpRequest [2] object. It allows web pages (using JavaScript) to perform asynchronous request to their originating server and fetch updated data from it. On the next step these data is used to update part of the web page information in timely and responsive manner without requiring page reloads. As the XMLHttpRequest object has support for transferring data in the XML format (presentation/view neutral encoding) the corresponding technology was named AJAX [2] (Asynchronous JavaScript And XML). Hence it becomes possible to develop web applications that look and feel in a way very similar to the regular desktop ones and provide the user with similar usage experience and capabilities. Keeping the business logic on the centralized servers (usually cluster of servers), allows easier management, support maintenance, and upgrades. Also, new schemes of application delivery and usage becomes feasible as pay-per-use, application service providers, click-and-run (no installation is required), application delivered as a service, etc.

The first widespread web applications were Google Mail (http://www.gmail.com) and Flickr (http://www.flickr.com). Another application are web applications are Writely (http://www.writely.com) word processing application (re-cently March 2006 bought by Google), calendar – 30 boxes (http://www.30boxes.com), CalendarHub (http://www.calen darhub.com).

## II. Motivation and Basic Requirements

In the field of web based spreadsheets applications the key players are NumSum (http://www.numsum.com), and iRows (http://www.irows.com), open source applications TrimSprea-dsheet (http://trimpath.com/project/wiki/Trim Spreadsheet), WikiCalc (http://www.softwaregarden. com/wkcalpha). Unli-ke the rest of the web applications, at the time of this writing (March 2006), web spreadsheets are not feature complete, built for specific purposes and non-customizable. Open source ones are mostly unusable and are actually just a proof of concept that the real applications.

Above observations have motivated us to build web based spreadsheet engine to solve our specific needs and to be able to extend and customize it as needed. The requirements for our spreadsheet system are:

1. It must be web based - many different people can use it from any physical location provided internet connection is available.
2. It must be component based – consisting of logically separated software units with well-defined interfaces and behavior.

[1] The work report in this paper has been partially supported by Project IIT-010061 "Technologies of the Information Society for Knowledge Processing and Management".

[2] Ivo Marinchev is with the Institute of Information Technologies, Bulgarian Academy of Sciences, Acad. G. Bonchev Str., Bl. 29A, 1113 Sofia, Bulgaria. E-mail: ivo@iinf.bas.bg

3. It must be loosely coupled – every component must be easily replaceable, and must not depend on the rest of the components as much as possible.
4. It must be scalable – can scale up and down with minimal changes.
5. It must support different user roles (groups) - some users can design and edit spreadsheets' definitions (designers). Regular users can just fill the data in the cells that are not locked.
6. Entered data must be kept separated from the spreadsheet definitions so that it can be reused in different spreadsheets and/or other related applications.
7. Ajax version must be able to work on IE and Firefox. If it is possible Opera and Safari must be supported as well.
8. The system must be able to import MS Excel spreadsheets and convert them into its internal form – reuse already created spreadsheets and use MS Excel as a primary design tool until comparable web based spreadsheet designer is developed.

## III. System Architecture

Fig.1 depicts the architecture of our system. It is compliant to the requirements enumerated above and, in practice, is 4-tiers (layers) system that comprises the following layers:

1. *User interface layer.* It consists of all of the software components that are transported through the network to the user system and executed on it. In this layer there are basically two type of implementations – Ajax based that run from the regular web browsers or rich client interface (for example JWS or Eclipse RCP). This layer communicates to the next layer using standard open communication protocols that have HTTP transport (so that the clients can be used behind firewall).

2. *Web layer.* It consists of software components that are executed in the web server environment. These components can be CGI scripts, PHP pages, Java servlets, JSP pages, etc. or even web services implemented in any programming language.

3. *Application layer.* It represents the main application business logic. It comprises all algorithms and internal data structures that are involved in spreadsheets management and processing

4. *Persistent storage layer.* It is a set of databases (Relational and/or XML) that organize the information on the persistent media.

In practice, in case of small deployment scenarios, it is possible the Web layer to be merged with the Applications layer. For example we have such situation when the application layer is implemented using web services (SOAP or REST) that are consumed directly from the user interface layer. This possibility of layer merging is a key feature of any systems that is projected to scale up and down.

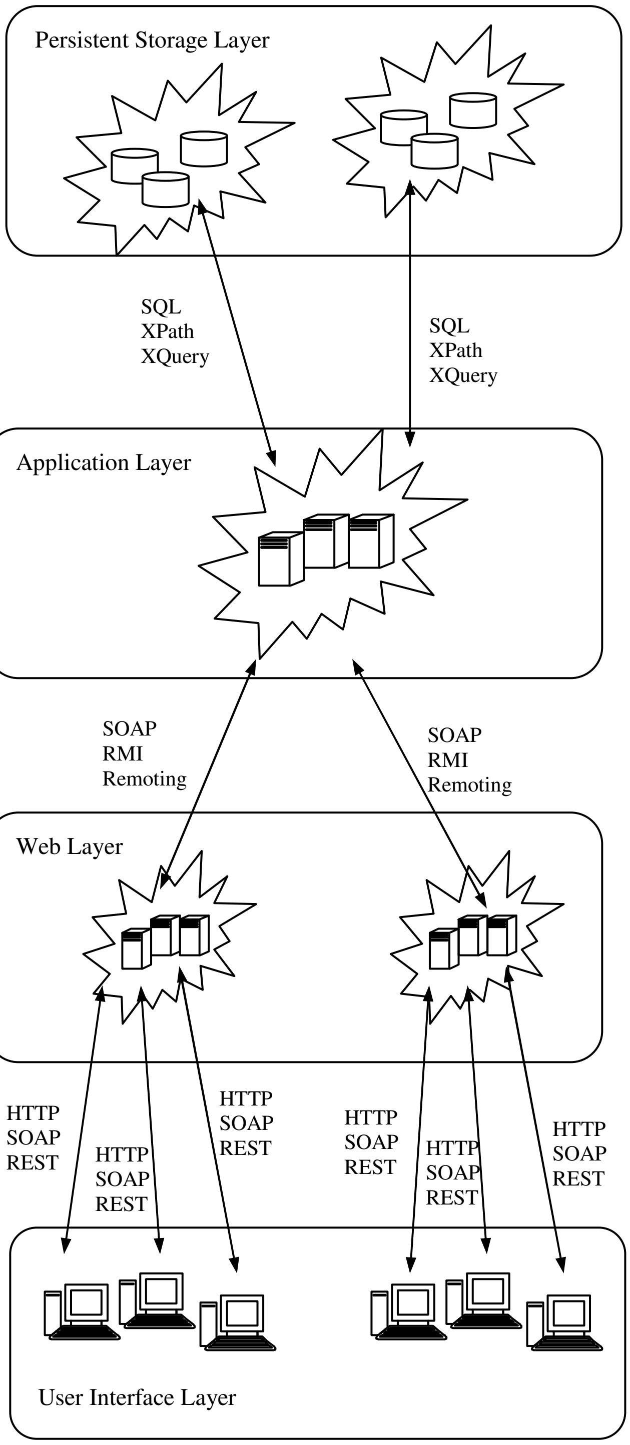


Fig. 1. System architecture.

As Fig.1 shows, all system's layers are very loosely coupled and communicate only with open and standard complaint protocols. This feature makes the system to be completely agnostic to hardware and software platforms that execute its components. In practice any client part and any server part can be executed on different operating system allowing almost any available hardware to be used as a client or a server.

## IV. System Implementation

The current version of the system is implemented as a standard 3-tier web application. Technical specifications are as follows:

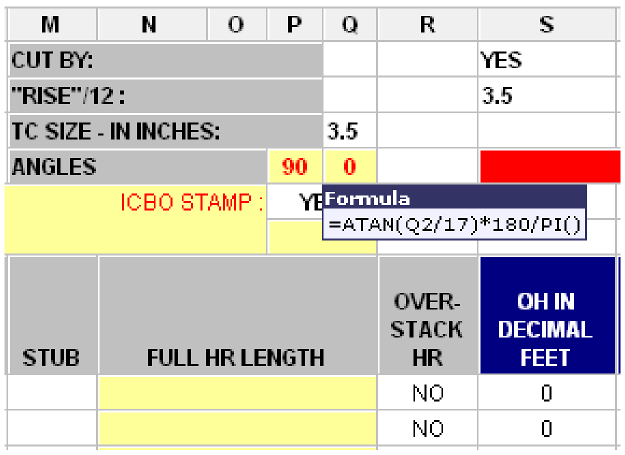
1. *User interface layer*. JavaScript library that uses Ajax requests to update and fetch data from the web layer. It works on IE (5.5, 6.0), Firefox (1.0, 1.5) and to some degree on Opera (8.5). It is possible to build Java Web Start client in the future.

2. *Web layer*. At the moment it consists of PHP pages (convenient only for stateless services) and Java servlets (better scalability, applicable for stateful services that use big data models). Current implementation uses REST [1] services. Although REST is not a standard but an architectural style, its light-weighted, requires fewer resources, and is simpler and faster for quick-and-dirty implementations. Later we can convert inter-layer communications to the complete SOAP, WSDL, WS-I stack if it is needed. This layer contains all of the algorithms and data structures that dynamically build user interface pages (screens) of the system. It is also responsible for getting and validating request parameters from the user interface layer and reformatting the response data if it is needed.

3. *Application layer*. In the current deployment scenario this layer is merged with the web layer because there is no need for separated web and application layers in the small size deployments. This layer contains all of the algorithms and data structures that implement the core business logic. Upon client requests (coming from the user interface layer) it fetches the data from the persistent storage layer and builds internal data model of the manipulated spreadsheets. Then it uses these data models to re-calculate the spreadsheets values based on the user changes and sends the updated data (user changes) back to the persistent storage layer and forth (recalculated fields values) to the user interface layer.

4. *Persistent storage layer*. Uses relational database storage. It was tested with MySQL and SQL Server, but as it uses standard SQL queries it should work with any complaint relational database system.

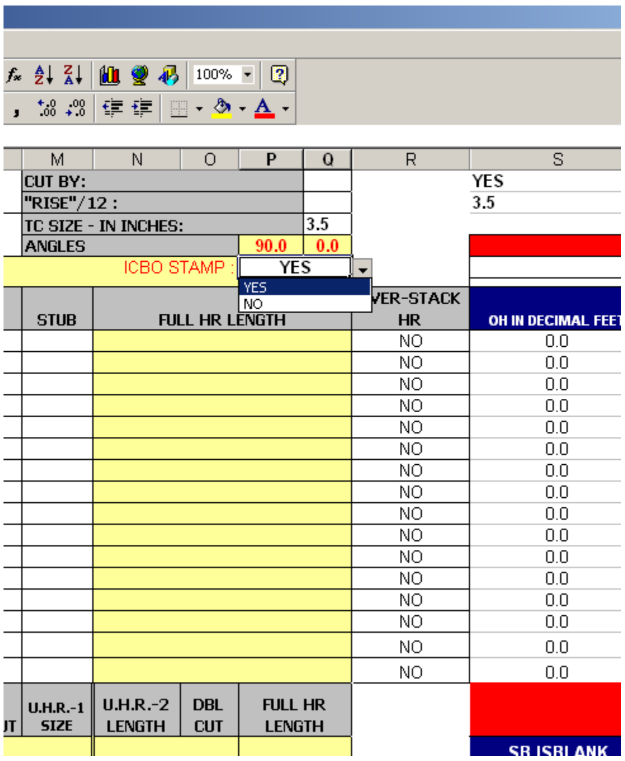
Figures 2, 3, and 4 depict a visual demonstration of our system. Fig. 3 shows a complex spreadsheet opened within MS Excel that we used for testing purposes. Fig. 4 shows a screenshot from the same spreadsheet converted to the format of our system loaded in it and then opened in Internet Explorer. Fig. 2 shows one of the unique features of our system its - ability to show the formula as tooltip when the mouse hovers over the corresponding cell.



| M | N | O | P | Q | R | S |
|---|---|---|---|---|---|---|
| CUT BY: | | | | | | YES |
| "RISE"/12 : | | | | | | 3.5 |
| TC SIZE - IN INCHES: | | | | 3.5 | | |
| ANGLES | | | 90 | 0 | | 🟥 |
| | ICBO STAMP : | | YE | | | |
| STUB | FULL HR LENGTH | | | | OVER-STACK HR | OH IN DECIMAL FEET |
| | | | | | NO | 0 |
| | | | | | NO | 0 |

Formula
=ATAN(Q2/17)*180/PI()

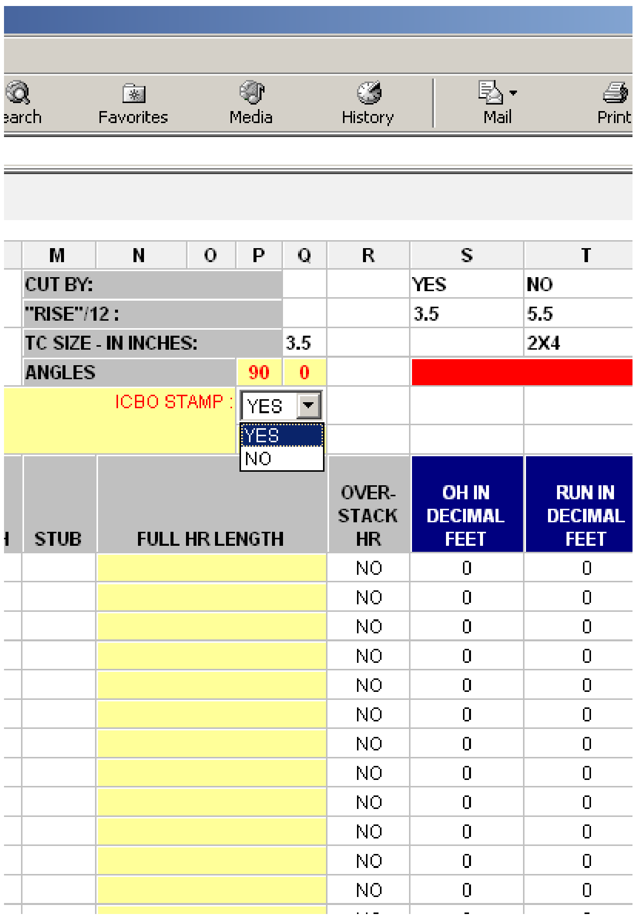
Fig. 2. Some of the new features in our system – shows formulas as tooltips



| M | N | O | P | Q | R | S |
|---|---|---|---|---|---|---|
| CUT BY: | | | | | | YES |
| "RISE"/12 : | | | | | | 3.5 |
| TC SIZE - IN INCHES: | | | | 3.5 | | |
| ANGLES | | | 90.0 | 0.0 | | 🟥 |
| | ICBO STAMP : | | YES | | | |
| STUB | FULL HR LENGTH | | | | VER-STACK HR | OH IN DECIMAL FEET |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| | | | | | NO | 0.0 |
| JT | U.H.R.-1 SIZE | U.H.R.-2 LENGTH | DBL CUT | FULL HR LENGTH | | 🟥 |
| | | | | | | SB ISBLANK |

YES
NO

Fig. 3. Complex Spreadsheet in Excel



| M | N | O | P | Q | R | S | T |
|---|---|---|---|---|---|---|---|
| CUT BY: | | | | | | YES | NO |
| "RISE"/12 : | | | | | | 3.5 | 5.5 |
| TC SIZE - IN INCHES: | | | | 3.5 | | | 2X4 |
| ANGLES | | | 90 | 0 | | 🟥 | 🟥 |
| | ICBO STAMP : | | YES | | | | |
| STUB | FULL HR LENGTH | | | | OVER-STACK HR | OH IN DECIMAL FEET | RUN IN DECIMAL FEET |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |
| | | | | | NO | 0 | 0 |

Fig. 4. The same spreadsheet from Fig. 2 in IE

## V. CONCLUSION AND FUTURE WORK

In this paper we presented our flexible architecture for development of scalable, and customizable web based spread-sheet engines. We presented also our concrete implementation of this architecture. Below we itemize some of the improvement and additions that have to be implemented in near future in order to make the system applicable in wider task scopes:

1. Improve the support of some of the alternative browsers as Opera and Safari that currently do not work due to bugs in their DOM implementation. Opera works with simple spreadsheets but not with our complex one.
2. Implement more of important MS Excel features that are still missing.
3. Add some unique features as the ability to use CGI scripts or web services in formulas. One example of this is getting weather conditions, exchange rates, interest rates, stock quotes, etc. in real-time.
4. Many other optimizations and improvements.

## REFERENCES

[1] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD Thesis, University of California, Irvine, 2000
[2] AJAX, http://en.wikipedia.org/wiki/AJAX
[3] CSS1 Specification, http://www.w3.org/TR/REC-CSS1/
[4] CSS2 Specification, http://www.w3.org/TR/REC-CSS2/
[5] ECMA-262, ECMAScript (JavaScript Specification), http://www.ecma-international.org/publications/standards/Ecma-262.htm
[6] W3C Document Object Model, http://www.w3.org/DOM/