Calculation Enabled Thin-Client Architectures and Extensions

Joseph Pally[1], Salih Yurttas[2]
[1]ZCubes, Inc.
10777 Westheimer, Suite # 808
Houston, TX 77042
USA
joseph@zcubes.com

[2]Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
USA
yurttas@cse.tamu.edu

ABSTRACT: *Among the architectural organizations for server, client, or hybrid configurations that can be adopted for application designs, newer cloud-based designs have been adopting a predominantly server based approach in the recent years. Though this approach correctly assumes that the fantastic improvements in server capacity in terms of memory, processing, and bandwidth, it also completely ignores the massive capacity and scalability of client machines in satisfying user needs. The traditional client architectures use the binary capability of the client, which is generally not oriented towards the web. The extensive dynamic capabilities of the browser allow for platforms of infinite extensibility and omni-functional power to be delivered via the net. This paper considers the adoption of intelligent immersive omni-functional technology with special focus on calculations towards creating standards-compliant documents that can be intermingled with semantic web structures. The paper documents experiences with such developments, and the implications in adopting such technology as a standard aspect of web-pages.*

Keywords: Omni-functionality, Websheets, Calculations, Intelligent documents, Immersive documents

Received: 11 November 2009, Revised 29 December 2009, Accepted 8 January 2010

1. Introduction

Though programs use intensive calculations internally, most pieces of software are unaware of calculations at the user interface level. Integrated calculations that are embedded into the functionality, has been sorely lacking even in reporting software, where we would expect calculations to be pervasive.

On the other hand, productivity software and other document creation software have been segmented; along with the data formats that lack any kind of standardization. Since 1991, the rise of the internet has created the possibility of document standardization. At the moment, web page formats (such as HTML and XML) have matured enough to provide good encapsulation for data, format and layout. Additionally, scripting has expanded even the primary scope of web pages from presentation and layout to logic.

The authors document a new elegant approach called "omni-functionality" in this paper that points to a variety of possibilities, in which new calculation based document structures can be architected. This is positioned against server-client architectures commonly adopted on the web for calculations by most online software vendors, as well as pure client solutions provided by the non-web-oriented vendors.

The authors will also introduce a practical and successful implementation based on the concept, as well as attempt to define what an intelligent immersive document is. The focus of this paper will be on the calculation abilities provided by such document structures – implying that every webpage can become more computationally powerful as provided by any known conventional spreadsheet software.

The calculation-enabled omni-functional document surpasses all conventional spreadsheet abilities including cell referencing, expression evaluation, multiple sheet/table cell organization, calculation engine, recursive evaluation, that is triggered upon requirement to ANY tabular element of a generic webpage.

This has the effect of bringing in the ability to do spreadsheet referencing and equations (like =A1+B1*SUM(TABLE2!A1:B3)) in webpages, as well as calling remotely =XML("SouthKorea.rdf") to parse and provide further presentation of the data). The expressions are available not just within intra-document spreadsheet-like interfaces, but also the rest of the document-scripting (which even conventional spreadsheets have issues providing without accessing the Application object). Also, a Websheet calculation platform seamlessly interacts with the vector graphical elements and pictures (and other content) on the page, which brings in the ability to conduct calculation without limits. Also, flexibility of calculating using any data type, including pictures or arrays (not just numbers or text), indicates depth of calculation currently impossible in any commercially available spreadsheet. This also differentiates from server-based on-line spreadsheet tools, which simply attempt to move the calculation engine to the server-side from the client application, and provides no fundamental advantage to the next generation of calculation tools.

To take things further, omni-functional calculation platform described here handles advanced features like vector-charting, multi-dimensional (pivot) analysis, lower-end databasing, etc. within a simple webpage without server-contributions. By appropriately manipulating the design of a webpage, Websheets approach enable any webpage (and potentially all webpages) to become a calculation medium that exceed the power and flexibility normally available only in expensive pieces of fat-client calculation software currently sold in the market.

The point though is not just whether Websheets make calculation mediums better than spreadsheets, but that websheets enable documents that provide ALL common functionalities better (like drawing, presentation, composition) at the same time. This is the central appeal of omni-functionality.

Several attempts at creating web-oriented computation spaces have appeared such as grid computing (Fox 2003), web-services (Halpin 2006), etc., but they also are marked by architectural rigidity due to the inflexibility associated with the server-based and/or fat-client/applet architectures. Hence, in this paper we focus on the client-side architectures with thin-web-browsers. Further, this architectural approach scores over server based technologies due to the flexibility in functionalities that are inherent. For example, the ability to handle varied custom data types creates the possibility of computing with images, vectors, etc., as against just text and numbers.

2. Background

A compound document is considered to be a document primarily of text based content, which is intermixed with complex non-text elements like video, audio, pictures, etc. The earliest known platform to implement compound documents is the Xerox Star workstation in 1981. Software component frameworks such as OLE/COM and ActiveX (Microsoft), Java Applets, KParts (KDE), Bonobo (GNOME), and OpenDoc (Apple) use the idea of software components to provide compound document solutions. Primarily, the activation of a specific component is based on the data type of that component, and is achieved by invoking the application that specializes in performing that function. For example, activation of an embedded Excel chart launches Microsoft Excel behind the scenes; and Microsoft Excel then provides the functionality required to edit/modify/delete that component.

Such architectures sometimes create unstable and heavy combinations of software during runtime. Since the secondary software that gets activated within the context of the first is unaware of the process space, its contribution to functionality is limited to the strict data sharing as per the component interface standards. For example, if Excel is activated within Word, its context is quite restricted to the specific Excel embedded object and not to the rest of the Word document. Major disadvantages of isolated activation include (a) the inability of two embedded objects to interact with each other smoothly, (b) slow initial activation of embedded program logic, (c) general software instability and (d) limited possibility to conduct recursive embedding.

These issues are not limited to productivity software, but many other pieces of software that create documents that deal primarily with drawings, movies, images, etc. This is a major issue for software users, who would prefer documents to be like the way they think. HTML/SGML that is commonly used in web browsers do use EMBED tags to achieve similar effects as in documents. However, this is limited to view-only interaction between the data elsewhere and the embedded objects.

A classic document – whether it be a report, a presentation, a book, a paper, a notice – require elements of any data type in any sequence as the user desires and requires to express his/her ideas. In most cases, users use two-dimensional documents tuned for paper-formats (or Adobe PDF or similar) that users can print out and read – not to interact or to provide an immersive experience.

Web pages have been a departure from this trend, though for the most part two dimensional static presentations (with sparse highly customized logic) seem to be the norm. Online providers of functionality such as Google Docs have attempted to make web into more intelligent providers of functionality. However, here too the trend is to isolate such functionality into pockets of logic that are driven at the server. True free interactivity within a document is still missing.

Still, embedding other data-types into files of a certain type is still dependent on the features provided by the software. Web browser has somehow bucked the trend in having limitations on what can be embedded conveniently, by providing a high degree of freedom and flexibility. However, web browsers have remained primarily browsers, and not editors – contrary to Tim Berners-Lee's original vision (Berners-Lee 1999). Editable web is still rare, except for wikis and blogs which are custom made for editing (Iorio 2005).

Tim Berners-Lee's combining of the hypertext, Graphical User Interface and TCP/IP net was a giant leap in the way documents could be structured. This has made possible documents that may compose themselves out of resources strewn all over the Internet. Even then, conventional software have adopted document technologies like containers that hold everything, rather than a lighter structure – with associated resources combined on the fly as required.

The use of HTML aware client and server-side technologies have made it the most prevalent of content presentation and layout format. The ability of accompanying Javascript (and similar scripting languages like Visual Basic) to represent logic and dynamic action makes it far richer than most other formats, which try to store data alone in documents.

Conventional productivity applications have supported scripting techniques (such as Visual Basic for Applications that is used to script Microsoft Office Applications). These have been typically used to provide isolated and additional functionality to the specific application that host the script. On the other hand technologies such as OLE Automation Server have provided allows other applications to script an application. Techniques including CORBA (Common Object Request Broker Architecture), .NET Remoting, RPC (Remote Procedure Call), SOA (Service Oriented Architecture), ROA (Resource Oriented Architecture)/REST (Representational State Transfer**)**, etc., provide remote method invoking functionality to applications that can be used to beckon services that are housed out-of-process or out-of-machine. Another interesting technique is to use real-time messaging architectures such as TIBCO and Jabber (XMPP) to achieve interactivity among other processes that may be inside or outside the machine.

Effectively, the amount of data interactivity (and logic interactivity) among programs that are typically used to create conventional documents today (Boyer 2008) depend on a variety of inter-process communication, messaging, program specific scripting, etc. This makes effective omni-functionality complex in current mono-functional application architectures (i.e., applications such as word processor, painting tool, etc. that focus on one or few features).

A typical user would prefer not to have to worry about data formats or specific programs; and instead would prefer to achieve whatever they desire to do. With a segmented software landscape, glued together at best using complex and unstable technologies, such desires remain a pipe-dream.

3. Intelligent Documents

Most of the popular pieces of software have been organically developed over the last 20 years, by teams and companies that have never had a reason to work together. This has been the primary cause for segmentation, though efforts like Apple Hypercard and Microsoft Binder had attempted to solve this problem earlier with limited success. The missing ingredients may have been a universal platform, a universal language, a universal network, a calculation and graphical framework and a universal orchestrating or scripting language.

3.1 Characteristics of Intelligent Document

An intelligent document can be visualized as a document structure and platform that is: (a) active, (b) immersive, (c) interactive, (d) web-compliant, (e) seamless, (f) application-agnostic, (g) data-agnostic, (h) omni-functional, (i) shareable, (j) viewable, (k) editable, (l) function-agnostic, (m) feature-deep, (n) calculation-aware, (o) graphic-aware, (p) standards-based and (q) infinitely extensible (Pally and Yurttas 2009).
The following expands the intent of the terms as used above.

1. An active document allows the user to manipulate its data.

2. An immersive document allows the user to achieve all required functions without leaving the platform.

3. An interactive document allows the user to interact with the content during the lifetime of the document, through scripting if required.

4. A web-compliant document depends primarily on open web-standards or commonly available web-technologies at all stages of its lifetime.
5. A seamless document is a document that is not fragmented into pieces based on separate features, unless the fragmentation is for convenient deployment.
6. An application-agnostic document does not restrict its interpretation to a certain specific application only.
7. A data-agnostic document can contain any type of data that the user wants to deal with.
8. An omni-functional document provides any type of functionality to any part of the document in any sequence whatsoever.
9. A shareable document can be viewed and interacted among appropriate actors separated by time and/or space at the appropriate level of content granularity.
10. A viewable document is a document that provides view-only interaction during specific stages of its life-time.
11. An editable document is a document that provides editing capability to a document at specific stages of its life-time.
12. A function-agnostic document is a document that allows any function to be applied to any part of the document in any sequence based on the desires and needs of the user.
13. A feature-deep document is in which the functionalities provided are of sufficient depth that for a majority of uses, the available functionality encompasses the needs comprehensively.
14. A calculation-aware document is a document that can achieve a vast majority (if possible all) of calculations achieved normally through specialized applications like spreadsheets as a part of the document itself.
15. A graphics-aware document is a document that can create and manipulate vector and raster graphics of extreme complexity in multiple-dimensions.
16. A standards-based document uses common web and other standards to represent information, rather than a proprietary format that may be specific to a vendor.
17. An infinitely-extensible document is a document that can take new data and functionality at anytime during its lifetime for needs and uses that may be unexpected before or during its construction.

It would be desirable for such document structures to be equally malleable during construction, modification, viewing, and extension (i.e., while being extended to handle newer functionalities as requirements change).

3.2 Web Pages vs. Documents

Since the advent of the web, web pages and sites have been treated as isolated pages that are hyperlinked as required. Very limited functionality above presentation was expected out of web pages from the early days. Even today, much of additional functionality is delegated to the server, as can be seen in most online productivity tools.

Web pages and user generated documents have been seen as separate and independent ways of presenting and storing information – one primarily for the web and the other when you want to adopt some required functionality for a non-web user (as in intranets, emails, local networks, etc.). It is interesting to note that SGML (from which HTML and XML was later derived) was developed to manage law-office information at IBM. It needs to be recognized that HTML is equally (or even better) suitable for the creation, modification, transmission, storage and retrieval of documents for which we use older formats.

3.3 Storage Format Standards

There have been several attempts among vendors and bodies to adopt XML based standards for document storage such as OpenDoc (from Apple), OOXML (from Microsoft) and Open Document Format (ODF) (from OASIS, Organization for the Advancement of Structured Information Standards). However, these only define storage and interoperability among software clients. Still, the desire of the user as creator, editor and viewer of a document to interact with the information in a seamless way is still not met.

An open standard on storage may reduce the risk of specific vendors to dominate an application domain because they control the format. However, that does not solve the ever present problem of myriad of function points and interfaces to manage and achieve user intent.

That is, uniform formats does not enable seamless interfaces that does not require the user to decide *a priory* what interface to start with, and run into roadblocks when the user intent changes.

The lack of standardization in expression of logic and calculations in the different pieces of software has been another major issue. Spreadsheet software standards have been the most popular to express calculations. Mathematica (and other symbolic packages), Matlab, AutoLISP, etc. have had some success in this regard like spreadsheets, though these have been applicable only within their own specific applicable areas. Also, generic expression of logical structures has been evasive.

3.4 Mono-Functionality/Omni-Functionality

The most notable aspect of conventional software is the focus on one or few pieces of functionality. This is generally driven by the way that the software was developed, as well as the marketing messages that need to be adopted to reach its market.

Microsoft Office, Open Office, Google Docs, etc. are examples of Mono-Functional solutions that are put together into "integrated" solutions. The most interaction that is available among these solutions is still limited to copy/paste and object embedding.

Omni-functionality introduces the concept of the availability of any functionality that the users require – without regard to the sequence of creation or complexity. Whether the user wants to calculate or draw or compose, to any extent or complexity, the authoring platform seamlessly moulds itself to suit the user's requirements.

3.5 Infinite Functionality

With Dynamic HTML (DHTML) and Javascript being widely adopted by most major browsers, it has become possible to implement most of the functionalities using a browser. Overall, the ability and efficiency of popular browsers to (a) script the document object model as well as complex logic, (b) render media (such as images and video), (c) handle complex multi-layered layouts, colors, filters and vector graphics, (d) asynchronous and synchronous communication, (e) embed or include Flash and similar techniques, (f) insert dynamic scripts at run-time, (g) call web-services and (h) provide secure server-client communication has created the possibility to achieve infinite and seamless functionality.

3.6 Real-World Implementation

Starting in 2006, Joseph Pally and his team at ZCubes based in Houston invented a series of techniques that enabled a thin-browser to achieve a variety of functionality using Javascript and DHTML, with minimal amount of server contribution (Pally 2009). Over the years, the platform has achieved a plurality of functionality that includes extensive graphics and calculation abilities. As a comparison, conventional online or offline spreadsheets contain about 500 calculation functions, whereas ZCubes calculation functions exceed *many thousands* (CALCI Websheets).

However, the most interesting aspect of the observations is the seamlessness of the functionalities. For example, all functions that is normally provided in spreadsheets are available to the graphical or editing aspects of the software.

The infinite extensibility of this architecture can be noted in the fact that new script logic can be added to the platform during runtime. This is similar to add-in concept in conventional applications, however, the additional script logic can be accessed from anywhere in the Internet as and when needed.

Users have been using this interesting implementation to draw art classics to make models of the temperatures inside the earth. The platform can transform from greeting card software to website creation software to video wall creator seamlessly, without disturbing the user.

The ability to immerse the user in a platform that browses the internet as well as calculate without regard to whatever the user wants to do is indicative of the power of the web browser platform directed appropriately.

4. Immersive Calculation Spaces

As described, providing simple web-pages the ability to support extensive calculation functions has the side effect of substantial implications. At the moment, most calculation requirements that a page requires is provided by custom code. For example, in a webpage that shows mortgage loan depreciation, the software programmer has to create or include the code that conducts calculations regarding interest, amortization, etc. and then hard-wire the appropriate display controls. This obviously is non-optimal, time-wise and operationally. Since non-standard code is strewn all over, errors and incompatibilities abound.

The answer lies in the provision of calculation functions as *omni-present aspects* of web-pages. Since due to security reasons, API calls are restricted within a browser. Browser plugins are difficult to deploy and consume. Hence the safest delivery mechanism seems to be based on efficient javascript – which is a very powerful and expressive language that is universally understood by all serious browsers.

CALCI includes *all* functions conventionally available in spreadsheets and other mathematical tools. These functions can be called and launched from spreadsheet-like interfaces, or from any part, control, or logic contained in the page itself. The universal ability of a page to achieve standard calculations reduces the possible incompatibilities among web-pages. Few standard functions are available in javascript, and most of those that are available in spreadsheets are not.

For example, the mathematical functions such as SUM() of recursive arrays, or a financial function such as RATE(), or a date function such as WEEKDAY() do not work in javascript. For each of such functions, further programming is required.

The solution is to provide javascript with all conventionally available spreadsheet functions, as well as additional functions. The availability of these functions directly is useful, and ability to apply them in iterations, repetitions, and nested function calls should not be underestimated.

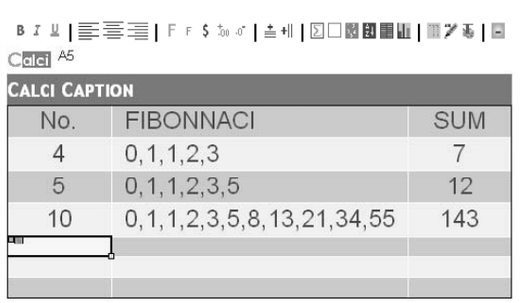
For example, the availability of a FIBONNACI () series or GEOMETRIC() series would enable web-pages that may support math related functions, while availability of COVARIANCE() may be of interest to the statistician. However, the overhead caused by providing these as default is very limited or negligible as witnessed by the implementation of CALCI. CALCI also introduces the concept of domain specific 'add-ons' that are activated as required by the webpage that gives domain specific calculations. For example, LOANS 'add-in' gives the webpage highly useful set of functions that relate to the payments and scheduling of LOANS.

The exhibited omni-functionality also enables the calculation modules to interact with graphics. For example, CALCI generates vector graphs that are interactive (as against image graphs generated by most online tools). Another side effect is the ability to generate vector graphics objects, like a hundred circles at increasing radii, etc.

6. Testing the Concept

In several examinations and testing sessions by faculty and students at universities and schools around the world, as well as enterprises of small and large sizes, the practicality and seamlessness of the Omni-Functionality concept that is exhibited by ZCubes, with special focus on CALCI Websheets, has been proven. In specific testing conducted by Salih Yurttas at Texas A&M University, the calculation capability of the omni-functional platform was considered, by intermingling with other functionalities, namely vector graphics. The sequential capability of the omni-functional platform was considered. The following contains arbitrary, natural and random sequences of functionalities demanded of the software upon testing (these are not prefixed set of tasks) to test the omni-functionality concept. Such sequences, though simple, require specialized websites or flash programs to conduct, and would entail quite a bit of programming. However, with the seamless omni-functional capabilities of ZCubes, this series of actions could be conducted without breaks or programming. The example had the following steps: (a) Draw a circle; (b) Calculate and generate circles of diameters that increase as in FIBONNACI series (pixels) at every step; (c) Locate the circles at points offset by random pixels each; (d) Logically connect to the circles, find their total areas; (e) Report the array of areas of the circles; (f) Report the total of the array of the areas of circles. By combining the graphical and calculation abilities of the platform, the sequences of actions were achieved.

Figure 1 shows the flexibility with which CALCI Websheets handle complex data types (other than simple text and numbers). The first column shows the input number, the second shows the FIBONNACI series that contain the input number of elements in the series. Note the fact that the elements in the second column are arrays, and not numbers. However, the third column is able to add the numbers in the arrays in the second column together. Here the calculations are done on the client-side, and not on server-side.


Calci A5

CALCI CAPTION

No.	FIBONNACI	SUM
4	0,1,1,2,3	7
5	0,1,1,2,3,5	12
10	0,1,1,2,3,5,8,13,21,34,55	143

Figure 1. Array and Series in CALCI

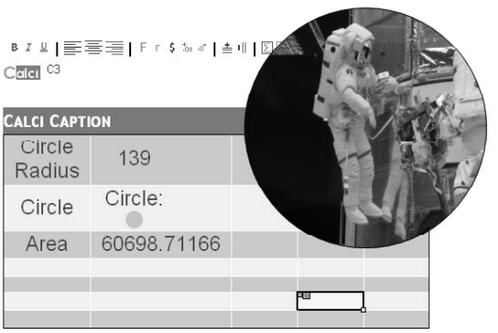


Figure 2. Circle Object and Methods

Figure 2 shows the referencing of an image contained within a vector circle of radius 139. The circle object is instantiated in the CALCI element, which is then used to calculate the area of the circle using B2.AREA() which gives 60698.71166. It is to be noted that the AREA function is part of the CIRCLE object, which is part of the geometry add-in. This hence also reflects the infinite extensibility of the calculation-ability. Add-ins to the CALCI platform are additional javascript modules that are dynamically loaded to the client, and all objects or modules is created on the client side (unless demanded by the object). With the new logic enabled by the incoming add-in, the calculations can then access properties or methods of the object or elements thus instantiated. This is demonstrated by the spreadsheet cell B2.AREA(), that uses the *Circle* object to determine the area.

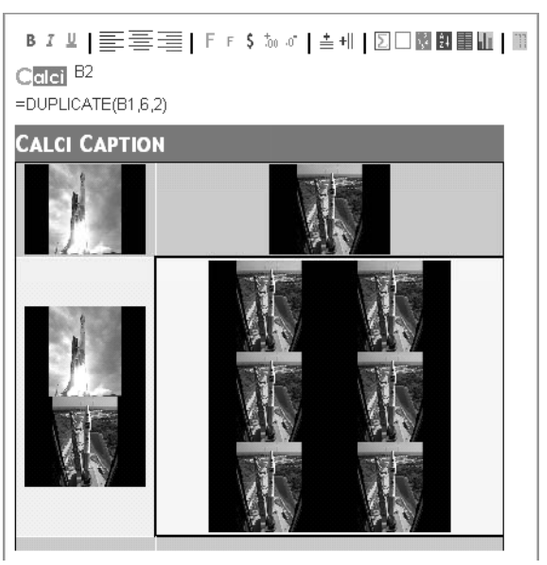
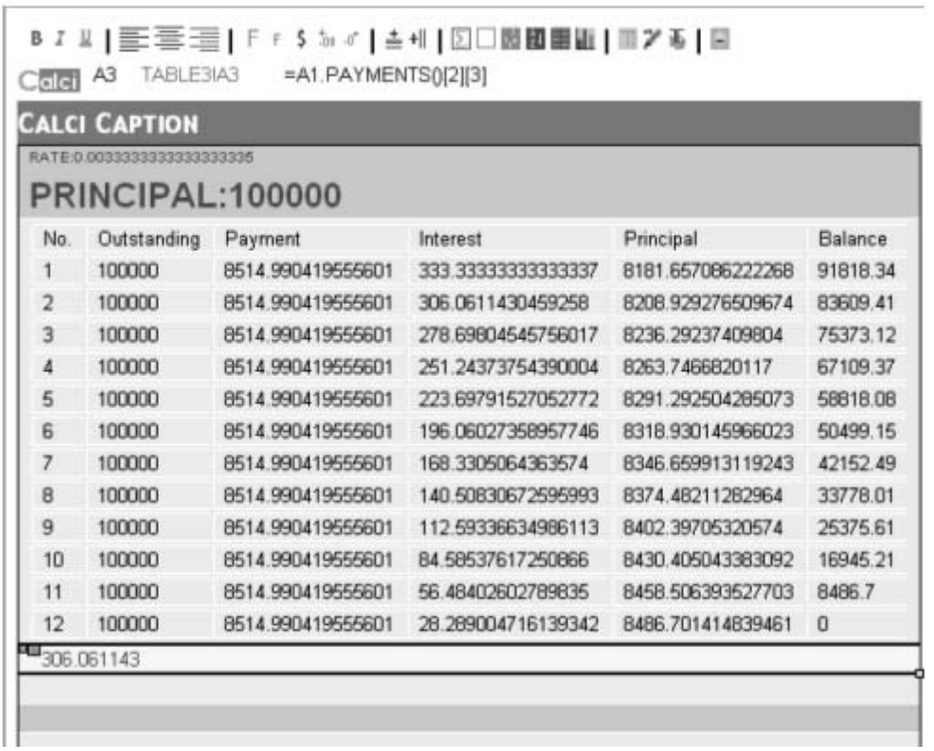


Figure 3. Picture Elements and Operations

Figure 3 above shows the flexibility of CALCI elements, in that the cells contain images. Cell A2 contains the equation A1+B1, which *adds the images* from cells A1 and B1. In Cell B2, the equation DUPLICATE(B1, 6, 2) duplicates the image in cell B1, 6 times in rows of 2. Clearly, these ar e not operations allowed or possible in conventional spreadsheets.

The notable aspect of Websheets is the availability of all spreadsheet functions in every web page, that can be used by the creator (or consumer) of the document to extend the logic (or try out different sequences of logic). Having the series of functionalities like the simple SUM() and COUNT() or the complex PEARSON() or NORMDIST() for every webpage to draw from itself opens up lots of opportunities and freedom on the Internet platform.

In the tests, the LOAN add-in logic was added to the omni-functional websheet. A loan was defined as =LOAN(100000, 4%, 12) , to represent a loan of $100,000 as the principal, 4% interest rate, and 12 payments (Figure 4). The interesting thing to note in this test case is the availability of the LOAN as an object for the further method calls to extract and present more



No.	Outstanding	Payment	Interest	Principal	Balance
1	100000	8514.990419555601	333.33333333333337	8181.657086222268	91818.34
2	100000	8514.990419555601	306.0611430459258	8208.929276509674	83609.41
3	100000	8514.990419555601	278.69804545756017	8236.29237409804	75373.12
4	100000	8514.990419555601	251.24373754390004	8263.7466820117	67109.37
5	100000	8514.990419555601	223.69791527052772	8291.292504285073	58818.08
6	100000	8514.990419555601	196.06027358957746	8318.930145966023	50499.15
7	100000	8514.990419555601	168.3305064363574	8346.659913119243	42152.49
8	100000	8514.990419555601	140.50830672595993	8374.48211282964	33778.01
9	100000	8514.990419555601	112.59336634986113	8402.39705320574	25375.61
10	100000	8514.990419555601	84.58537617250866	8430.405043383092	16945.21
11	100000	8514.990419555601	56.48402602789835	8458.506393527703	8486.7
12	100000	8514.990419555601	28.289004716139342	8486.701414839461	0

Figure 4. First cell shows the Loan with its properties. The second cell B1 shows the tabulated payments of this loan, interest, principal, and balance on the time-series

information. Also the second cell A2 contains =TABULATE(A1.PAYMENTS()) which specify that the results from the A1.PAYMENTS() which returns an array, which is then tabulated.

Note that cell A1 contains LOAN as an object that is instantiated within the omni-functional document space. The table is then rendered within the second cell. Here the cell contains a table — something other than a text or a number.

In this case, =A1.PAYMENTS() gives the interest payment of 306.061143 from the array. This shows that the results of the complex object can further be manipulated.

The availability of full-fledged computation in a free-space which is data-agnostic, function-agnostic, scriptable and graphics-aware indicates the intelligence and immersivity of documents possible with such omni-functional platforms.

7. Summary

The ability for web-based document concept that achieves omni-functionality with immersive calculation abilities is significant. The usefulness of universal availability of standard spreadsheet like function calls in web-pages (along with additional appropriate interfaces), which would allow for the creation of interactive intelligent documents that can have accurate and easy expression of logic and processing, is enormous. By keeping the interface functions similar to standard spreadsheet language, we open up the possibility of standard ways of requesting and providing functionality. At the moment, every custom implementation rebuilds each required function, which creates an increasing lack of standardization and increasingly complex systems. Apart from the provision of functions, the spreadsheet like functionality that parts of the entire document transforms to, creates a totally different level of interactivity and computation. The ability of intelligent immersive omni-functional documents to calculate can have enormous implications to reporting, analysis, forms, etc. The most prevalent platform of today – the Internet Browser – may eventually be considered as a virtual paper or medium, of immense power, flexibility and expressivity.

References

[1] Fox, G (2003). Grid Computing Environments, *Computing in Science and Engineering*, v.5 n.2, p.68-72, March 2003.

[2] Halpin, H (2006) One Document to Bind Them: Combining XML, Web Services, and The Semantic Web, *Proceedings of the 15th International Conference on World Wide Web*, Edinburgh, Scotland, May 23-26, 2006.

[3] Berners-Lee, T (1999). *Weaving the Web*, Harper, San Francisco.

[4] Iorio, D. A. Vitali, F (2005). From the Writable Web to Global Editability, *Proceedings of the 16th ACM conference on Hypertext and Hypermedia*, Salzburg, Austria, September 06-09, 2005.

[5] Boyer, J. M., et al., (2008). An Office Document Mashup for Document-Centric Business Processes, *Proceeding of the 8th ACM Symposium on Document Engg.*, Sao Paulo, Brazil, September 16-19, 2008.

[6] Pally J. Yurttas, S (2009). Intelligent Immersive Omni-Functional Documents, *Proc. 1st International Conference on Networked Digital Technologies (NDT 2009)*, Ostrava, The Czech Republic, July 29 - 31, 2009.

[7] Pally, J (2009). *The Thinking Things*, Silkrays, Houston, TX.

Authors Biographies

Joseph Pally is the CEO of ZCubes, Inc., a high-technology company based in Houston, and leads several software companies. He is a graduate of the Indian Institute of Technology, Madras, India, as well as Texas A&M University, College Station, Texas, USA. He is a well-known software scientist, and has been credited with the invention of ZCubes and many other advanced software technologies. He is also the author of two popular books – "Fail Fast, Move Faster," a book on a winner's attitude and "The Thinking Things," a history of computing spanning several millennia in short story format.

Salih Yurttas Doctor Salih Yurttas holds a PhD in Computer Science from Ege University, Turkey. He has been on the faculty of Texas A&M University, Department of Computer Science 1982-1991 as Visiting Assistant Professor and 1991 to present day as Senior Lecturer. He taught undergraduate and graduate courses at Texas A&M, Turkey, and People's Republic of China. His main interest is Programming Languages Design and Implementation for Large-Scale software development. He has developed reusable, modular language collections in Ada, C++, Java, and several imperative languages.