



WHITE PAPER : WEB APPLICATIONS UNPLUGGED

THE WORLD'S FIRST AJAX IDE THAT BRINGS
WEB APPLICATIONS TO THE DESKTOP AND
TAKES DESKTOP APPLICATIONS TO THE WEB

October 2005

Copyright © 2005 Morfik Technology Pty. Ltd.

Version 1.2

All rights reserved. Morfik and its respective logo is a trademark or registered trademark of Morfik Technology Pty. Ltd. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.

Morfik Technology Pty. Ltd.
P.O. Box 337
Hobart
Tasmania 7000
AUSTRALIA

WWW.MORFIK.COM

TABLE OF CONTENTS

Abstract	1
A new approach	2
Limitations of a page-centric architecture	2
The Morfik solution to limitations of a page-centric architecture	3
Limitations of HTML page layout	3
The Morfik solution to limitations of HTML page layout	3
Limitations of JavaScript	3
The Morfik solution to limitations JavaScript	4
Client-side handling of browser requests	5
The Morfik solution for client-side handling of browser requests	5
Database	6
The Morfik solution for database needs	6
Productivity	6
Appendix A – Multi language support in Morfik	7

Abstract

In less than two decades since its humble beginnings, the World Wide Web has not only permeated a fair portion of our lives but has also become the subject of much discussion and speculation as a viable alternative to traditional platforms for business applications.

For many years the web community has tried to overcome the limitations of web browsers and their related internet protocols by extending the capabilities of browsers and servers in a variety of ways. Some have tried to extend the functionality of browsers using plug-ins and applets, while others have tried to push the entire computing task to the server.

If taken to either of these extremes, the status of the browser is essentially reduced to that of a "dumb terminal". With applets and plug-ins, the browser is merely a host for another application which runs inside a box. With server-side computing, the browser simply displays what it receives like a slide-show.

While centralized systems, such as corporate server farms or applications hosted by ASPs (application service providers), are believed to need little more from the browser than the functionality of a dumb-terminal, in real life the browser has steadily grown in functionality to the extent that it is now a viable alternative platform for both the web and desktop applications – irrespective of their on-line or off-line state.

Recently, developments such as Asynchronous JavaScript And XML (AJAX), have captured the imagination of developers worldwide and have allowed us to re-examine the capabilities that have existed in web browsers for some time. Applications such as Google Gmail, Google Maps and Flickr have shown that the user experience in a browser can rival that of desktop applications, with the browser powered by nothing more than JavaScript, HTML/XML and XMLHttpRequest.

However, for AJAX to succeed in the long term, it must be supported by professional development tools that are specifically designed for creating web applications and also incorporate the design methodologies and features offered by products such as Visual Studio® and Delphi®. Ideally, developers should be able to leverage their existing language skills and use a familiar Integrated Development Environment (IDE) to develop AJAX applications without the need for learning JavaScript or for hand-coding HTML.

This is precisely what Morfik's WebOS AppsBuilder offers. It allows programmers to implement the business logic of their application in a high-level object oriented language of their choice and develop the presentation layer of their application using a visual design environment. WebOS Apps Builder compiles the project's code, resources and objects into an AJAX application that can run both on-line and off-line.

A new approach

"Our objective is to bring web applications to the desktop and take desktop applications to the web"

Can we use the browser as a platform for real-life business applications?

Will such applications provide a user experience similar to conventional desktop applications?

Will these applications work both on-line and off-line?

The answer to all these questions is a resounding yes! The capabilities required for this have been natively available in browsers for some time, yet we have been hampered by the difficulty of writing JavaScript code and distracted by inadequate attempts to extend the browser's functionality.

By combining the following elements in a single stand-alone package, web applications can be brought to the desktop and desktop applications taken to the web:

1. **HTML** - for the user interface
2. **JavaScript** - for the application logic
3. **XMLHttpRequest** - for asynchronous browser requests
4. **Web Server** - for handling the browser requests
5. **Database** - for managing data

This concept is not new and appears deceptively simple, yet its implementation presents enormous technical challenges. Unless these challenges are addressed and overcome, the concept shall remain on the drawing board.

Morfik technology has identified and addressed these technical challenges and in the process has created a unique Integrated Development Environment for developing business applications on the browser platform.

The following is a technical account of these challenges and their solutions.

Limitations of a page-centric architecture

Traditional web applications are comprised of a number of web pages that are separate from one another both spatially and computationally.

Unlike the user interface of conventional desktop applications, the browser content changes in a disjointed fashion – sometimes with unacceptable time-delays. This spatial separation of web pages has a detrimental effect on the user experience. Recent developments, such as AJAX, have proven successful in enhancing the user experience, and applications such as Google Gmail and Google Maps are examples of this.

Those who develop for the web are forced to scatter the business logic across the application space and find workarounds for managing the application state in an otherwise stateless environment. This computational separation of web pages has a detrimental effect on the reliability and scalability of the application. In contrast, conventional software engineering emphasizes a unified computational model to ensure reliability and scalability.

One option is to push all of the computing to the server-side in order to unify the computational space and make the application more reliable. But this has three undesirable side-effects: first, it further limits scalability; second, it is dependant on both the availability of band-width and the reliability of the connection; and third, the application is no longer available when it is unplugged.

The Morfik solution to limitations of a page-centric architecture

WebOS AppsBuilder applications are not page-centric. The browser content “morphs” according to the requirements of the application. This simplifies state management and works hand-in-hand with an application-specific AJAX engine. WebOS AppsBuilder creates this AJAX engine from the business logic written in a high level language of choice using its unique and patent pending JavaScript Synthesis Technology (‘JST’). This approach unifies the computational space of the application across the server and the client.

WebOS AppsBuilder applications do not use applets, plug-ins or cookies. The result is an application comprised purely of HTML and JavaScript, which rivals desktop applications in user experience and computational integrity.

Limitations of HTML page layout

HTML pages are designed to display their content in a fluid layout much like a word processor. Fluid layouts take the shape of their container. Changing the size of the browser’s window will change the alignment of page components. Web designers have managed to find work-arounds to address many undesirable side effects of this fluid model.

With the advent of Cascading Style Sheets, browsers gained the ability to display the elements of the page in a predetermined fixed position – irrespective of the shape or size of the containing window. This fixed layout is similar to conventional desktop applications. But since neither the size of the browser window nor the resolution of the client display is controlled by the developer, the CSS model has some undesirable side-effects also.

The ideal model is a plastic layout that combines the strengths of fluid and fixed models. Such a plastic layout is particularly useful when database reports and tabulated data are displayed or page elements change their size and position due to user interaction or business logic.

The Morfik solution to limitations of HTML page layout

WebOS AppsBuilder’s plastic layout offers the strengths of both fluid and fixed layouts. In Morfik applications, the content of the browser is a hierarchy of heterogeneous nodes. Each node is aware of the existence, state and behaviour of other nodes and can respond to layout changes at run-time according to the rules set out by the programmer at design-time. In other words, at design-time, the programmer can define both the fixed position of each element as well as its run-time plasticity. This uniquely incorporates the best of both worlds in layout design.

Limitations of JavaScript

The difficulties of writing extensive and yet coherent JavaScript code is the Achilles’ heel of AJAX and if not addressed could eventually slow down its widespread uptake.

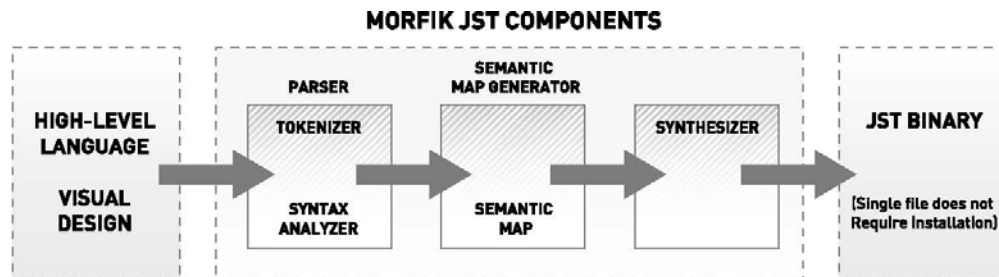
Few acknowledge or recognize that JavaScript’s capabilities extend beyond that of a mere scripting language for light programming tasks. However, market acceptance of JavaScript for implementing large-scale applications faces the following practical challenges:

1. Syntax - JavaScript borrows its syntax from C. While C/C++/C# and Java programmers feel at home with case-sensitive short-hand syntax, the large number of developers who have mastered other languages such as Visual Basic and Delphi face a frustrating transition.
2. Semantic Design - JavaScript is a prototype-based (instead of class-based) object-oriented language. This also presents developers with the challenge of adopting a whole new mindset.
3. Lack of rigor - JavaScript does not provide the developer with a rigorous programming model. For example, it does not support type declarations, nor can functions receive their parameters by reference, to name a few. This limitation is a disadvantage in large scale programming.

4. Interpreted - Interpreted languages do not offer the benefit of compilers in rigorous enforcement of application integrity at compile time. Small changes to an otherwise perfectly working program can cause unpredictable run-time crashes. For example the most common error messages in existing web-page scripts are "object expected" or "object does not support this property".

The Morfik solution to limitations of JavaScript

Morfik solves these problems with its revolutionary JavaScript Synthesis Technology ('JST'). Using WebOS AppsBuilder, programmers implement the business logic of their application in a high-level object oriented language of their choice (e.g. C++, C#, Java, Delphi). WebOS AppsBuilder then compiles this code into a JavaScript AJAX engine.



The process is a true compilation and avoids boilerplates or code snippet libraries. The source code written in the object-oriented, strongly typed language of choice is first passed through a parser that includes a tokenizer and syntax analyser. The output of the parser is passed through a semantic map builder which builds a detailed semantic map of the entire application. This map conveys the full "meaning" of the application logic. This semantic map is then compiled into JavaScript code that is semantically identical to the original code written by the programmer and conveys the exact same "meaning".

This process offers several advantages:

- Mastering a programming language is a long and arduous process. Switching to a new language requires much more than learning a new syntax. It requires an idiomatic shift in the programmer's thought processes. This skill takes time to acquire. By allowing developers to program in the language they have already mastered, Morfik not only encourages the uptake of AJAX and improves the quality of the output, but also reduces the development cost of AJAX applications.
- The clean separation of the parser, semantic analyzer and compiler allows Morfik to compile from a variety of source languages into a variety of target languages. The only condition is that the source language is either object-oriented (e.g. C++/C#/Java) or has been specifically modified to support object-oriented constructs. (See Appendix A for further details on) "multi-language support in Morfik".
- Strongly typed compiled languages follow a rigorous programming model that has made compilers the preferred tool for building large and complex applications.
- Compiled code is, by definition, more reliable and scalable than interpreted code. Although the output of the Morfik compiler is interpreted JavaScript, the compilation process nevertheless ensures rigorous type checking and enforcement of referential integrity in producing reliable and scalable applications.
- The AJAX engine created through the Morfik process is an exact semantic equivalent of the high level source code which reflects the programmer's intentions without imposing limitations. Code-snippet libraries and pre-fabricated component frameworks, by their very nature, limit the programmer's ability to choose the appropriate level of abstraction.

Since the output of this process is neither an executable in machine-code, nor a one-to-one translation of source code, nor a collection of predefined code snippets, WebOS AppsBuilder's process is referred to as JavaScript Synthesis Technology ('JST').

Client-side handling of browser requests

Browsers are designed to work with servers using a stateless request-response model. Normally, the browser and the server do not co-exist on the same hardware, and browsers need to stay connected to the web in order to maintain a dialog with the server. Of course, this is natural and the way of things on the web. However, to bring the web applications to the desktop and enable them to work after they are unplugged from the web, one must be able to handle the browser requests locally.

One obvious solution is to install an HTTP server on the local host. This will also fulfill the second part of the objective, namely taking desktop applications to the web. Many mature and stable examples of HTTP servers are freely available. Some are stand-alone applications whilst others are small embedded systems. Alternatively, the relevant subset of HTTP server protocols can be implemented within the application so it can natively communicate with the browser and process browser requests.

More importantly, the browser vendors have recognized the benefits and possibilities that local handling of browser requests can offer, and they are planning to add this ability to the browser itself – effectively merging the HTTP server and the browser into a unified system.

The Morfik solution for client-side handling of browser requests

WebOS AppsBuilder has an open and highly flexible architecture which can support the local handling of browser requests in a variety of ways. To provide a mature, stable and well tested option that can allow its applications to run under Windows, Linux and OS X, WebOS AppsBuilder tightly integrates an embedded open-source Apache server into Morfik applications by default. This is not an exclusive option and will not limit the developer's choice of other methods.

Database

Business applications are predominantly data-driven and depend heavily on Relational Database Management Systems (RDBMS). In the corporate world, data is centralized and database servers do not normally co-exist with the client applications on the same hardware. Consequently, the client needs to stay connected to the network in order to maintain a dialog with the server.

To bring data-driven web applications to the desktop and enable them to work after they are unplugged from the network requires a local database engine.

This will not only fulfil the second part of the objective, namely taking desktop applications to the web, but also will facilitate the implementation of large scale distributed databases and distributed computing.

Many mature and stable examples of RDBMS are freely available. Some are stand-alone applications whilst others are small embedded systems.

The Morfik solution for database needs

WebOS AppsBuilder has an open and highly flexible architecture and can integrate local database engines or provide remote database connectivity in a variety of ways. To provide a mature, stable and well tested option that can allow its applications run under Windows, Linux and OS X, WebOS AppsBuilder tightly integrates an embedded open source Firebird (formerly Interbase) RDBMS server into its applications by default. This is not an exclusive option and will not limit the developer's choice of other database engines.

Productivity

AJAX applications have created a lot of excitement but until professional Integrated Development Environments become available, developers will find it difficult to justify the effort – and therefore the cost – associated with AJAX programming.

It is Morfik's belief that designing an IDE around code libraries and component frameworks is a repetition of past mistakes. Such tools make it deceptively easy for developers to get simple applications up and running quickly only to have them hit the wall as their projects grow in size and scope.

To ensure reliability, scalability and maximum productivity, JavaScript Synthesis Technology leverages the power of high level languages and the existing skills of programmers.

WebOS AppsBuilder is a world class system which incorporates the features that are required for a truly professional AJAX-based Integrated Development Environment.

Appendix A

Multi language support in Morfik

At the core of the WebOS AppsBuilder Integrated Development Environment for AJAX lies the time-tested and proven technique of using a strongly typed object-oriented language and a compiler, which strictly enforces referential integrity, high levels of modularity, re-factoring and polymorphism within the source code, to develop applications that are reliable, scalable and easy to maintain.

The output of a compiler is invariably in a language different from that of the source code – usually machine code but sometimes an intermediate language. A good example of this is the Microsoft compiler that takes the source code in C#, VB.NET or J# and produces code in Microsoft Intermediate Language (MIL). When examining a snippet of sample code in Microsoft Developers Network (MSDN), one is usually given the option of seeing it in one of these languages. Yet if these code snippets are carefully compared, one realizes that semantically all three languages are identical! Therefore, the MIL code generated is also the same.

To achieve this, in the case of Visual Basic, Microsoft had to take it to the next level, and developed the strongly typed object oriented VB.NET. Although unpopular with some VB programmers this move was nevertheless a technical necessity.

Morfik has used this strategy in developing Morfik Basic, Morfik C#, Morfik Java and Morfik Object Pascal (with more to come) for the AJAX platform. Programmers choose their favourite syntax (or even mix and match a number of them) to develop their application. This allows them to benefit from the rigorous software engineering techniques and system design methodologies inherent in the object-oriented programming paradigm. WebOS AppsBuilder then synthesizes the source code into semantically equivalent JavaScript code and packages it into an AJAX application. In this model, the synthesized JavaScript in WebOS AppsBuilder is the equivalent of MIL in .NET

In addition, WebOS AppsBuilder does not prevent extending the JavaScript output by linking-in external hand written JavaScript code libraries. However, such external code would not benefit from the same rigorous checks as native WebOS AppsBuilder code.

WebOS AppsBuilder has a flexible and highly modular architecture. There is a clean separation of the syntax parser, semantic analysers and the target code synthesizer. This enables WebOS AppsBuilder to compile from a variety of source languages into a variety of target languages. The only condition is that the source language must be strongly typed and either object-oriented (e.g. C++/C#/Java) or has been specifically modified to support object-oriented constructs.

Creating a new Morfik source language is an easy task. For example Morfik Basic was developed in 2 months by only one engineer. As for the target language, should there be a future demand, the WebOS AppsBuilder synthesizer can even be modified to create its output in MIL.

Of course one can choose to develop complex AJAX applications such as Google Gmail and Google Maps by hand-coding the entire systems in HTML and JavaScript. This can even be done using “Notepad” but if history has given us one valuable lesson, it is that using the most appropriate tool to improve productivity and quality is an economic imperative. In a highly competitive market where users demand more complexity for less cost and in less time, developers will find themselves out in the cold if they fail to leverage their existing skills and find reliable tools to improve the quality and timeliness of their efforts.