WHITE PAPER

Wakanda: Blending Traditional Developer Productivity with Modern Web Architecture and Standards

Sponsored by: 4D

Al Hilwa

October 2011

IDC OPINION

Application development as practiced today is known to require heavy investment in detailed technical knowledge, causing professional developers to sink years of learning and practice to achieve proficiency in language or system. Thus, any technique or set of techniques that can reduce the proficiency ramp-up time for application developers is sure to be well received. Dynamic languages have evolved to provide higher levels of abstraction and productivity as have model-driven approaches to application development that take advantage of graphical composition for the construction of the user interface or the data model. Leveraging the widely available base of Web application developers who are skilled in building standard browser-based applications is another simpler idea. The Wakanda application platform takes advantage of all of these techniques and adds three other important areas of productivity enhancement that make it particularly productive and appealing to no-nonsense developers looking to build architecturally modern applications efficiently and quickly. These capabilities are:

- An end-to-end offering that provides development tools, front-end runtime and back-end server, and data tiers that are integrated and designed to work congruently
- An abstract object-oriented data model for back-end programming that reduces impedance mismatches between relational database and application code
- A unified JavaScript programming language syntax for both frond-end and back-end development that leverages the broad familiarity of this language among Web developers

IN THIS WHITE PAPER

In this white paper, we explore the use of JavaScript on the server, also known as server-side JavaScript (SSJS), and examine the new application development platform known as Wakanda, which employs JavaScript, Web standards, and modern object-oriented techniques with the traditional value proposition of end-to-end, integrated application development.

Global Headquarters: 5 Speen Street Framingham, MA 01701 USA P.508.872.8200 F.508.935.4015 www.idc.com

TABLE OF CONTENTS

SITUATION OVERVIEW

Today, the application development space is in the midst of significant transformations that have accelerated as a result of important changes in front-end devices and back-end computing architectures and business models. At the front end, the arrival of the smartphone has given rise to new ways for users to interact with technology through touch-oriented interfaces on highly mobile and sensor-equipped devices. The richness of context that such sensors provide (e.g., geolocation information, compass, camera, microphone) and the granularity of applications available for mobile devices are changing how consumers use technology in deep and broad ways. The changes have begun to hit businesses through both the "bring your own device" movement, where cell phones are selected and purchased by employees, and a new generation of tablet form factors that are derived from smartphones. These changes are promising to completely overhaul how the front ends of new applications are built and distributed through consumer or enterprise app stores. At the same time, business model changes in IT organizations have continued to offload more and more infrastructure and application management to external entities through outsourcing. Additionally, these changes have stimulated more granular service offerings where more and more software is being offered "as a service" to incrementalize expenditures. A variety of technology trends in software and hardware, such as Moore's law and virtualization, have given rise to these changes, but the net result of the changes is that application architectures are in the throes of rapid transformation.

A BRIEF HISTORY OF APPLICATION ARCHITECTURES

From the early days of computing through the 1970s, computing was characterized by centralized back-end computers that were accessed by primitive text-based devices called terminals. At that time, applications were architecturally monolithic, where all components ran on the same machine, usually inside a single address space. Early computing was centered on the server. In the 1980s, the arrival of PCs and local area networks (LANs) in business settings gave rise to business application architectures in the form of dBASE (1982) and FoxPro (1984). These applications relied on a passive file-server engine for data sharing and concurrency control, forming the beginnings of distributed application architectures and moving the center of gravity for new business applications almost entirely to client devices (PCs).

The 1990s: The Emergence of the Web

In the 1990s, we witnessed the emergence of increasingly more sophisticated client/server architectures that relied on both server and client logic collaborating and communicating. A variety of styles of client/server architectures were introduced by various platforms, including multitier architectures that introduced additional server tiers. By the end of the decade, the server-centric Web architecture had become the standard for writing new applications. The Web architecture was based on a publishing and hyperlinking model that relied on static Web pages composed on servers but displayed on graphically capable devices through browsers. Thus, in the early Web, the center of gravity for network computing was much closer to the server.

The Past Decade: Doubling Down on Web Architectures

Throughout the first decade of the 2000s, Web architectures evolved considerably as dynamic HTML began to be used to extensively animate Web pages and support more complex server and client processing. By the end of the decade, Web architectures reached an apogee where complex JavaScript client/server communication using Asynchronous JavaScript and XML (Ajax) became mainstream and plug-ins like Flash were used for graphically complex applications. This shifted the pendulum back again to where the center of gravity of application logic was much closer to the client. In the current decade, application logic has continued to become more complex and varied on both client and server, and client and server platforms themselves are evolving at considerable speed. Client devices are rapidly diversifying beyond the PC, and browser technology is becoming more important than ever as investment in the Web is increasing. On the server, the trend to move processing off-premises to run on more systemized cloud architectures has made it essential to invest in application modernization to bring older applications into Web architectures. Despite the birth of new proprietary application platforms catering uniquely to either client devices (e.g., iOS and Android) or cloud architectures (e.g., Amazon AWS, Microsoft Azure, Force.com), the investment in Web architectures has continued. IDC expects Web applications to grow in importance on mobile platforms and to eventually constitute a significant share of all mobile applications. Mobile and Web applications will leverage cloud application platforms on the back end.

THE RISE OF DYNAMIC LANGUAGES

While programming languages number in the hundreds or even thousands, they can be broadly classified into families that share common characteristics and programming models. Programming languages can be subdivided according to the different characteristics that they share. One important division is between imperative and declarative languages, the former composed of lists of instructions and the latter expressing computation largely without specifying control of flow. Thus, markup languages such as HTML, database query languages such as SQL, and functional languages such as LISP are regarded as declarative, while most familiar languages such as C/C++ and Java are considered imperative. Another major division is between languages that use static typing systems and those that use dynamic typing. While statically typed languages are considered the bulwark for large complex systems because of their readability, dynamically typed languages often provide higher levels of productivity, though for simpler tasks. Of late, language evolution has meant that many dynamically typed languages have become extremely adept at dealing with more complex problems, while their often interpreted or just-in-time compiled nature provides a more appealing style of development. The improved sophistication of integrated development environments (IDEs) has meant that most of the readability shortcomings of dynamic languages can be significantly reduced.

THE WEB ECOSYSTEM

At some level, programming languages, as well as their frameworks and tools and the developers who practice them, operate like natural ecosystems where developers move more smoothly within than across. Over the past 15 years, an ecosystem of Web technologies and developers has evolved around a broad spectrum of programming languages, frameworks, application platforms, and development tools revolving around the construction of Web sites and, more recently, Web applications. This ecosystem has grown in recent years to become larger than any other ecosystem IDC has examined, encompassing the entire range of dynamically typed languages. The languages that thrive in a particular ecosystem typically solve specific application problems in a particularly effective way. In the Web ecosystem, several languages such as PHP and JavaScript have reached superstardom, while many other languages and technologies play prominent roles (e.g., Ruby on Rails and the Spring framework). PHP and JavaScript are the two technologies that show up at the top of most surveys of developers. While these two languages are often used together as halves of the logic of a Web application, the two differ significantly in their programming models. PHP is typically used for server code, and JavaScript is used for client code. Recently, the concept of running JavaScript on the server has been gaining currency in the Web ecosystem, and a number of frameworks and technologies have emerged to support this approach. IDC believes that with the right adjustment to the programming model, a single language used across both the client and the server parts of an application can simplify the development process, providing a productive and unifying approach that can reduce development project complexity and shorten application delivery time.

WHAT IS JAVASCRIPT?

JavaScript was invented by Brendan Eich at Netscape to help inject computation and animation into the static pages of the early Web. The technology, which was internally known as Mocha, was first officially named LiveScript in the September 1995 beta of Netscape Navigator 2.0 and then renamed JavaScript as a result of a joint announcement between Netscape and Sun Microsystems (now Oracle). The name recognized the similarity of syntax between Java and JavaScript but otherwise has been a perennial source of confusion because beyond superficial syntactic similarities, JavaScript in fact is markedly different from Java in many areas, including its object model and type system. Microsoft shipped its version of the language, which it called JScript, in August 1996 as part of Internet Explorer 3.0. The ECMA standards organization took on the task of specifying the language more formally, releasing the first ECMAScript standard in mid-1997 with the aim of ensuring that implementations of scripting in browsers were compatible. Since then, the use of JavaScript has mushroomed in a variety of places, such as for scripting inside PDF documents and as ActionScript, which is the scripting language of the Adobe Flash runtime. JavaScript was soon complemented by other standards such as DOM that began to enable the dynamic Web before the end of the 1990s. The language continued to evolve, supporting more sophisticated and complex capabilities that have led to ever more dynamic Web sites, especially with the help of Ajax, which provides communication between client and server through the JavaScript XMLHttpRequest object. Such network communication is most often conducted with JavaScript Object Notation (JSON) syntax. Ajax is credited with transforming the Web

from Web sites to applications. Together with the standardization and broader implementation of the set of technologies known as HTML5 (including CSS3, Web Socket, etc.), JavaScript now plays a pervasive role in Web applications as almost all the major HTML5 capabilities require scripting.

THE ROOTS OF SERVER-SIDE JAVASCRIPT

In the earliest days of the Web, when client devices possessed limited processing power, servers performed all Web page composition and manipulation, shipping the whole Web page for display as a response to every browser request. Server-side scripting was the key technique that enabled the venerable Common Gateway Interface (CGI) architecture to provide server manipulation and computation through operating system code. In those early days, operating system scripting languages such as Perl were the most popular. At the time, JavaScript played a relatively small role on the client, but in fact, it was present on the server as the LiveScript part of the Netscape LiveWire Web application framework that eventually shipped in Netscape Enterprise Server 2.0 in 1996. At the time, Microsoft also included JScript in its dynamic scripting technologies known as Active Scripting (e.g., Active Server Pages or ASP), which it rolled out in IIS 3.0.

Without specific frameworks for access and manipulation of server objects, files, and data, there was little need to use JavaScript on the server, and other languages, in particular PHP, evolved to take on the mantle of ever more sophisticated server-side scripting. JavaScript on the client and PHP on the server are likely the most common pair of languages ever used together to construct applications. PHP and JavaScript are different in many ways, however, and that difference has meant that developers who take on the task of end-to-end application development have had to cope with the differences in language syntax and object model of the two languages. The increased popularity of JavaScript and the number of developers skilled in it have led to efforts to make the languages more alike. Recent versions of PHP have added functional programming constructs similar to JavaScript's such as lambdas and closures.

The latest trends to deal with the complexities of end-to-end Web development have focused on using JavaScript itself on the server. Recently, new communities have grown around technologies to harness JavaScript on the server. Rhino, which is a Mozilla project that has continued to evolve the Netscape JavaScript implementation, is a popular choice at the heart of many implementations. The Node.js event-driven server framework has also received considerable attention and has been productized by Joyent. Additionally, the CommonJS API specification and community, which is driving a common specification around JavaScript outside the browser, has also been actively pushing the technology. By the beginning of 2011, a visible intensity around server-side JavaScript was observed in the industry.

WAKANDA: UNIFYING SERVER AND CLIENT

Wakanda is a new application platform being offered by 4D, a vendor of integrated database and development tools. The Wakanda platform takes advantage of 4D's long-running expertise in delivering highly productive and integrated application platforms to bring a new Web-based application architecture to developers. The Wakanda platform is

composed of three main components designed to work well together so that they can simplify and speed up the development of complex enterprise Web applications. Many characteristics of the Wakanda platform become clear upon examination of the various ingredients of this system. Key Wakanda components are listed in the following sections.

Wakanda Studio

The IDE for the Wakanda system includes the standard capabilities typically available in development tools, providing a client- and server-aware set of features that should contribute to the typically high levels of productivity usually derived from such integrated end-to-end systems:

- **Solution Manager:** This is the main starting environment that allows developers to view their applications, which can be aggregated into projects and solutions, as well as launch other components of the Studio IDE.
- **Datastore Model Designer:** The model designer functionality allows developers to visually specify the datastore structure (schema) and data attributes to be used for the solution at hand, as a set of data classes, in a highly productive, model-driven development style.
- **GUI Designer:** The GUI designer allows the graphical construction and manipulation of user interface widgets and their binding to server data elements.
- **Code Editor:** The editor provides support for HTML, CSS, XML, and JavaScript. Along with the standard formatting and syntax highlighting typically provided by editors, the code editor provides advanced capabilities such as type-ahead and awareness of server-side framework APIs leveraging Wakanda's integrated nature.
- **Debugger:** A server-side debugger for JavaScript provides the traditional capabilities of setting breakpoints and examining variables.
- **User and Group Manager:** This feature manages access control for authorized system users.

Client Runtime

The Wakanda Ajax Framework is the primary API available to developers in the Wakanda system. It includes several important subsets of APIs, which are generally used in an asynchronous fashion:

- **Dataprovider:** Low-level APIs allow for manipulating and controlling server data from the client side. They communicate with the server for data parsing and result-set caching. The dataprovider is mainly used on behalf of user interface widgets involved in display functions, but it can be used on its own as well.
- **Datasource:** This is a high-level API for the dataprovider that provides an event-driven model for programming data applications. Through the datasource API, subscriptions to back-end data can be established that allow automatic or manual updates (CRUD) of objects based on server or user events. Datasource

can also be used to bind widgets with pure client-side data (JavaScript arrays, objects). As a key capability, the datasource API enables the automatic synchronization of multiple widgets bound to the same datasource.

- **Interface Widgets:** User interface elements built with standard HTML5, CSS3, and JavaScript can be bound to datasources so that they can receive automatic updates from the server. The widgets can be easily customized and animated through the embedded jQuery framework or any additional libraries.

Wakanda Server

The Wakanda server communicates with the client runtime through JSON-based messages. All server components are administered through a Web interface requiring no specialized front end. The server supports the CommonJS module architecture, making server-side scripting easily extensible. As an example of this extensibility, mail support and a MySQL database connector are provided as parts of the system. The key differentiators for the Wakanda server architecture are the integrated and optimized datastore system and the SquirrelFish Extreme (JavaScriptCore) JavaScript just-in-time compiler adapted from the WebKit project.

- **Wakanda HTTP Server:** The server provides the basic processing for HTTP Web requests coming from the browser or other Web services–consuming systems. The integration of the server with the back-end datastore provides an efficient and optimized environment for data-oriented Web applications.

- **Wakanda Datastore Model & API:** Instead of providing a raw relational interface to the developer, Wakanda provides an entity model accessible programmatically by the developer through both native server-side JavaScript and REST-like HTTP APIs. The key advantage of this approach is simplified programming so that not only the data but also the model's business logic is made available to the client.

- **Worker:** Developers are able to utilize JavaScript multithreading capabilities according to standard W3C specifications. Web Worker and System Worker threads are supported. All aspects of the W3C specification that are meaningful on a server are supported.

- **File & Image:** Developers can work with the server file-system objects, including images of various supported formats that can be manipulated as data-class attributes.

- **Console Logging:** JavaScript logs can be produced and accessed by developers.

- **Security and Access:** This component is designed to work with Wakanda's self-contained user and group authentication and access subsystem.

- **Application:** This component provides miscellaneous classes for managing Wakanda server applications (or projects), including those that control the operations of the integrated Web server.

- **Socket:** Direct TCP/IP communication is allowed. XMLHttpRequest is available as a server-side API.

- **CommonJS.** This module architecture makes server-side scripting easily extensible. Mail support and MySQL database connector are two examples of pure SSJS modules.

WAKANDA'S DIFFERENTIATORS

Wakanda injects several modern concepts into the established idea of an integrated development and deployment environment. Such environments were popularized in the 1980s and early 1990s because of their proven track record in delivering developer productivity in constructing enterprise applications compared with the mainstay compiled languages in use at the time (e.g., COBOL). These integrated development and deployment systems were characterized by application systems built with scripting-style development languages (then known as 4GLs) authored through specialized text editors (the predecessors of today's IDEs) and managed by rich runtimes that surfaced database access methods. These systems came from many vendors that competed to introduce a variety of innovations over time. Wakanda preserves the integration aspects that such development and deployment systems offered, providing improved consistency, security, and maintainability across the entire application, yet utilizes modern standards-based tools, interfaces, languages, and communication protocols.

Integration

One of the key levers for improving ease of use for any system is the reduction of complexity produced by preintegrating components to ensure that they work together in an optimized fashion. In modern development projects, developers often choose industry-standard components and integrate them to produce a specific project. A developer would have to select a programming language, an editing environment for the code, a container environment for managing the server code, a database system to store the data, and other components, depending on the needs of the project. The value of the selection process is the best-of-breed nature of the components and the strategic safety of each component, which might be a market leader for its narrow space. The cost of using such systems is that the development team is saddled with the complexity of integration, requiring deep multifaceted skills brought together by different members. In an integrated system, many of these components are designed and tested together, as well as optimized to work together, saving considerable time and effort and reducing the range of skills required to produce efficient systems. Wakanda features multiple system components designed together to work together, promising significant productivity gains.

Standards Orientation

The downside of integration is often that components cannot be easily combined with other technologies. Wakanda avoids this downside with an architecture that is based on a modern Web paradigm where the back end and the front end can be used with non-Wakanda components through the use of standards-based technologies such as REST APIs, JSON for communication, JavaScript for scripting, and HTML5 for markup. The

result is that Wakanda applications can take advantage of interesting JavaScript client libraries such as jQuery and interoperate with other back ends as is often required in enterprise systems.

Open Source

The Wakanda designers did not stop at the use of standard and commonly used technologies; they also decided on an open source strategy to generate community and partner stakeholders that can share in the success of the system and continually evolve to meet changing needs. Wakanda components will use various appropriate open source licenses that will allow community members and partners to contribute improvements that are made available for all Wakanda users. Additionally, the Wakanda team has articulated an interest in supporting other server-side JavaScript platforms.

CHALLENGES/OPPORTUNITIES

The Wakanda application platform brings several compelling advantages to developers. Wakanda will appeal to development shops interested in powerful yet highly productive application platforms that produce applications that conform to modern Web architectures. Areas of challenge and opportunity for Wakanda include the following:

- **Platform maturity.** This is a general area of challenge encountered in the birthing of any new platform. Working through this phase requires active recruitment with early adopters and in-depth engagement by the Wakanda development team to provide the needed help. The early days will yield bugs, which the vendor would have to fix. A significant enabler for this phase is that Wakanda's owner, 4D, has a long-running track record in maturing application platform product releases that achieve solid stability and deliver differentiated developer productivity. This is a challenge that 4D is expected to work through successfully, and the most eager early adopters are likely to be longtime 4D customers that are aware of 4D's track record.

- **Interoperability.** This is a challenge for all end-to-end development platforms that are made of parts that are designed and optimized to work together. The question becomes whether applications can be written with different client or server frameworks that can interact with Wakanda's data model. Wakanda's reliance on standard technologies such as JavaScript and REST APIs makes interoperability highly workable and potentially not problematic. The Wakanda server runs on multiple operating system platforms, providing options that can accommodate changing requirements. Additionally, the open source nature of the various parts of Wakanda implies that investments in adapters or interface layers can be shared in the ecosystem. Nevertheless, this is an area that might require attention and investment from 4D, either by documenting the patterns and modalities of interoperability with other systems and frameworks or by partnering with others to implement requested connectors.

- **Cloud deployment.** Developers increasingly expect their application platform or framework to be available as a service through a cloud operator. Automated application deployment and management has become the defining criterion of application platform–provisioned services, and 4D has the opportunity to define a PaaS strategy to support its developers. This will expand the potential reach of Wakanda to a variety of systems that might be initiated by business groups or agencies working on behalf of smaller companies that might choose to deploy the resulting applications outside the firewall. Cloud deployment options are likely to attract both existing and new developers who are looking for frictionless deployment of new applications.

CONCLUSION

Wakanda is a well-designed end-to-end application platform that comes complete with a graphical development environment catering to modern developer requirements. Wakanda is endowed with clean and rich data interfaces on server and client, which support an object-oriented style of development that takes advantage of modern Web technologies such as asynchronous communication. The application development approach supported by Wakanda abstracts much of the complexity in developing enterprise applications rapidly and productively. Wakanda is further distinguished by its forward-thinking leverage of the popular JavaScript language for both client and server scripting, making it potentially appealing to a large segment of the developer community as well as minimizing the overhead and surface area of what has to be learned to become competent in the platform. Early adopters of Wakanda will have the privilege of working closely with its storied vendor, 4D, to bring modern Web applications to life in a more productive manner than they likely ever have before.