Wt: a Tool for Building the New Web

White paper

Koen Deforche <koen@emweb.be>, August 2009

The evolving web

While the web was initially designed as a distributed information storage, quickly it was reshaped to provide basic interactive functionality, through dynamic web pages. At that time, PHP and ASP.net were created as tools for the development of such dynamic pages.

With the gradual improvement of web technology, modern web browsers have become capable of providing superior interactivity through the use of AJAX. AJAX is the technology that allows updates to an already rendered web page based on information fetched from the web server in the back-ground. As simple dynamic web pages are evolving into complex interactive web applications, new tools are being developed to more easily create such interactive web applications. Wt (C++) and JWt (Java) are such new tools that take a radically new approach to designing rich web applications.

Another factor that has equally contributed to the improved web browsing experience today is the increased bandwidth available to most users (with ADSL, cable modems and 3G connections offering a many-fold increase over old PSTN modems), together with a major upgrade of capacity in the the internet back-bone. This higher bandwidth leads to a predictable and lower average network latency. Indeed, one can achieve a round-trip time < 130ms consistently for cross-Atlantic connections, and thus high performance web applications may offer an interactive experience that comes close to a native desktop application. Nowadays, slow web applications are caused by slow servers, rather than a slow network.

Challenges for complex web applications

Modern browsers have the technology to render applications that provide a highly interactive user experience. But this remains largely untapped because the development of such applications is complex and time consuming using the traditional tools such as PHP or ASP.Net, which were not designed with these uses in mind.

These traditional tools have problems addressing the complexity related to interactive web applications for several reasons, including performance, security, accessibility and the new trend of combining applications from different publishers in a single web page (Facebook apps, Google map or calendar widgets, ...).

Pages, sessions and performance

HTTP is a stateless protocol, and every request is in principle unrelated to any other request. To serve a coherent experience to a user, sessions were introduced which typically identify and track a user that has logged in, and is used to offer personalized pages.

The traditional tools are page oriented and the contents of each page is recomposed for every request, with the session information only used to keep track of the user. Contrary to what is often argued, this leads to bottle necks in the back office not only because the page and any logic that relates to it (such as access control) is recomputed, but also because all relevant data must be fetched again. In contrast with the 130ms cross Atlantic round-trip latency, this processing in slow scripting languages may easily take more than a second of processing time and back-end load, providing a sluggish web experience.

Extensions to these page based frameworks, such as *partlets* in Ruby on Rails or *update panels* in ASP.net allow to provide partial updates using AJAX in a rudimentary way, which breaks accessibility (for search engine robots and older browsers), and still requires that for every request, all information relevant for the user session (such as access control) is recomputed.

Security

Security is not to be taken lightly on the Internet. Stories about security breaches make frequent headlines (as recently as the credit card theft of 170 million on-line accounts, now in the news). Cross-Site Scripting (XSS) and SQL Injection are the most frequently abused vulnerability these days. Yet, current tools such as PHP and ASP.net offer no built-in protection and the burden for securing user input and output lies entirely with the developer. Dynamic scripting languages such as PHP have the additional disadvantage that user input can become interpreted as a PHP statement.

Not only XSS attacks are a threat, but for proper authentication, each request must be validated to determine whether the user is authenticated to execute the associated action or access the returned information. Again, traditional tools offer few automated procedures to secure applications and the developer is responsible for determining this by implementing the proper checks for each request. Because of Cross Site Request Forgery (CSRF) threats, cookies cannot be solely relied on for authenticating a user, and only a form of URL rewriting can be used, but this is rarely done in practice because it is tedious to implement.

Since security is usually an invisible part of the deployed web application, it is often sacrificed under time pressure at worst and simply labor intensive and tedious to implement and test at best.

Accessibility

Since web technology evolved and continues to evolve with small steps, all web browsers support a different part of the standards, and do so in slightly different ways. To make an application accessible to the broadest audience possible, traditional approaches require a choice to be made between accessibility, which means supporting only the common denominator, and interactivity. A highly interactive website that relies heavily on AJAX or other newer techniques will not be accessible for those who do not have browser support for it.

An important type of “browsers” without JavaScript/AJAX support are search engine robots. These need to navigate your web site to index its contents.

For some applications search engine optimization is not important (e.g. an on-line mail application does not have any public contents), and thus these will more easily take advantage of AJAX features. But for most websites, including community websites with public profiles, the trade-off usually favors accessibility instead of interactivity.

Social applications

The web is not only becoming a platform for web applications, but also becoming a place with *web operating systems* where multiple applications may run

simultaneously on a single page, interacting with each other.

The most visible examples of this trend are Facebook and OpenSocial applications: a 3rd party application may integrate within a community website and interact with social information as if it were a native part of the web site. This requires a fundamentally new approach to a web application since the application runs not as a single page, but integrates in another page, using only JavaScript. This differs so much from traditional page-based web applications, that there is in practice little opportunity for sharing of code between both types, while the functionality may in many cases be the same.

How Wt addresses the new needs for web applications

Wt has been designed to bring the desktop programming model to web application development, and at the same time address challenges for modern web applications. Rather than structuring an application as a sequence of pages, a Wt web application runs within a single page, and updates its contents based on user actions.

Unlike page-based frameworks, session information is stored in memory in the application server throughout the session life time. This is used to improve server processing load and application interactivity and to eliminate common security problems.

Whereas PHP was (and still is) a successful tool for developing dynamic web sites, Wt has been designed to simplify the development of **maintainable, interactive, secure** and **accessible** web applications.

Interactivity

When accessed with a modern browser, a Wt web application will automatically optimize rendering for best interactivity and use AJAX for all event handling, returning only to the browser that information needed to update the page based on an event.

When handling an event, after establishing its validity (for security reasons, see below), the application immediately jumps to the functionality that responds to the event within the same context as where it was when it last updated the display. In this way, the application server only performs the minimum amount of work needed, and because of the session state that is available to it, it may take advantage of any relevant state for the user that was previously computed. This causes Wt applications to be highly interactive, have a pleasant user experience (in which the network latency is typically the highest factor), and generate a server and back-end load that is 10 to 100 fold lower than traditional script based solutions (like PHP, Ruby, ...).

The implementation of Wt has been optimized for performance, and web applications are well usable even when deployed on low-power ARM devices (180MHz) with limited caches.

Security

Wt uses its session state and knowledge of the user interface to protect the application automatically against many vulnerabilities:

- XSS is mitigated by only allowing Wt itself to generate JavaScript, but not content added by the user. This content is always filtered against malicious tags, while preserving normal XHTML formatting. This built-in protection could not be built into PHP because at any point in the page PHP needs to allow the insertion of JavaScript statements.
- CSRF is mitigated by not relying on cookies for session authentication (while cookies may still be used to preserve login information across sessions). This

does not lead to ugly URLs since the application stays within the same page, and only changes the named anchor part of the URL.

- Application logic is automatically verified: only requests corresponding to events that are exposed on the current page are allowed, and any other request is automatically blocked. In this way, a user cannot break free from the implicit context of requests that are allowed by what he sees on his screen.

Accessibility

While Wt is certainly not the only tool for adding AJAX capabilities to a web application, it is the only tool that addresses accessibility problems in a transparent way, while maximally taking advantage of AJAX (for every request!) to cut down on communication needs and server load. Wt examines the capabilities of every client, and adjusts its rendering strategy accordingly.

When accessed using a user agent that does not support AJAX (such as a search engine robot), Wt will automatically revert to plain HTML post-backs for reacting to events. This built-in graceful degradation requires no intervention from the developer since he only states what should be updated in the user interface in response to an event, but not how.

Within an AJAX session, Wt will keep the same page and use named anchors to simulate browsing through internal pages (e.g. http://website.org/#user/koen) For a user this has the same effect since he can navigate back and forward and set bookmarks to revisit later.

At the same time, Wt will also generate and react to the equivalent real paths (http://website.org/user/koen), and these URLs are used by plain HTML sessions and search engine robots.

Social applications

Since Wt makes abstraction of how a widget is rendered, a Wt application can easily be adapted to be an OpenSocial application. Wt may be used to define a *WidgetSet mode* application, which manages one or more widgets (a widget is a self-contained building block of a page, rather than a full page) inside another page. Wt's widget set mode uses the same kind of techniques to provide a tight integration and interaction within the host page, as used by OpenSocial to embed applications in an OpenSocial container.

This allows the reuse of any part of a Wt web application as an OpenSocial widget, provided that it interacts with the OpenSocial API to retrieve user profile information.

Conclusions

- New browser capabilities, increased Internet bandwidth and lower latency bring the potential for highly interactive web applications.
- Although frameworks originally designed for developing dynamic web sites may in principle be used to develop such web applications, there are performance, security and accessibility problems inherent to their design.
- To address this new scope for the web as an application platform, there is a need for new development tools, such as Wt. When using Wt, a developer can focus on the contents while the framework takes care of security and accessibility, while offering a superior user experience and vastly lower server load.