AJAX(Asynchronous Java Script and XML)

Pramod kumar & Ruchi Yadav
Dept. of Information & technology, Dronacharya College of Engineering
Farruhknagar, Gurgaon, India
Email:ruchiyadav477@gmail.com

Abstract

In recent years, information system based on browse/server architecture (namely B/S architecture) received more favor by enterprises. Ajax technology consists of five parts. They are HTML, JavaScript, DHTML, DOM and XML. With the help of cooperation and collaboration of these technologies, they can optimize the conventional enterprise information system by using an asynchronous way. Meanwhile, a quickly-responded and smoother user interface was provided. Enterprise information system with Ajax can be operated in a more efficient way, which means even use the current hardware, it can provide more load capacity, be more stable and serve more clients in parallel. In this paper: we present two kinds of information system models, one use conventional B/S architecture and the other use Ajax enhanced B/S architecture. First, we build both of the systems in accordance with typical business applications (search files, database access, etc.). Second, we use standard web pressure test tool such as Microsoft Web Application Stress tool to test both of the systems to get information like concurrent user number and average response time. Finally, with those experimental data, I compare and found out the difference between the two systems. The results presented in this paper propose a good way for enterprises, to enhance the information system performance, capacity and stability under a definite hardware facilities circumstance.

Keywords:

Ajax; HTML; DHTML; XML; DOM

1. INTRODUCTION

Over the course of the past decade, the move from desktop applications towards web applications has gained much attention and acceptance. Within this movement, however, a great deal of user interactiveness has been lost.Classical web applications are based on a *multi page interface* model, in which interactions are based on a page-sequence paradigm. While simple and elegant in design for exchanging documents, this model has many limitations for developing modern web applications with user friendly human-computer interaction.Recently, there has been a shift in the direction of web development. A new breed of web application, dubbed AJAX (Asynchronous JavaScript And XML), is emerging in response to the limited degree of interactivity in large-grain stateless Web interactions. At the heart of this new approach lies a *single page interface* model that facilitates rich interactivity. In this model, changes are made to individual user interface components contained in a web page, as opposed to refresh the entire page. Thanks to the momentum of AJAX, single page interfaces have attracted a strong interest in the web application development community. After the name AJAX was coined in February 2005, numerous frameworks and librarieshave appeared, many web applications have adopted one or more of the ideas underpinning AJAX, and an overwhelming number of articles in developer sites and professional magazines have appeared. Adopting AJAX-based techniques is a serious option not only for newly developed applications, but also for existing web sites if their user friendliness is inadequate.A software engineer considering adopting AJAX, however,is faced with a number of challenges. What are the fundamental architectural differences between designing a legacy web application and an AJAX web application? What are the different characteristics of AJAX frameworks? What do these frameworks hide? Is there enough support for designing such applications? What problems can one expectduring the development phase? Will there be some sort of convergence between the many different technologies?Which architectural elements will remain, and which ones will be replaced by more elegant solutions?Addressing these questions calls for a more abstract perspective on AJAX web applications. Despite all the attentionthe technology is receiving in the web community, there is a lack of a coherent and precisely described set of architectural formalisms for AJAX enabled web applications. In this paper we explore whether concepts and principles as developed in the software architecture research community can be of help to answer such questions. In particular, we propose SPIAR, an architectural style for AJAX applications, and study to what extent this style can help in addressing our questions.This paper is organized as follows. We start out, inSection 2 by exploring AJAX, studying three frameworks (Google's GWT, Backbase, and the open source Echo2) that have made substantially different design choices. Then, in Section 3, we survey existing architectural styles (such as the Representational State Transfer architectural style REST on which the World Wide Web is based), and analyze their suitability for characterizing AJAX. Next, in Section 4, we propose SPIAR, describing the architectural properties, elements, and constraints of this style. Given SPIAR, in Section 5 we use its concepts and principles to discuss various open issues in AJAX frameworks and application development.We conclude with a summary of related work, contributions,and an outlook to future work.

2. AJAX FRAMEWORKS

2.1. Ajax

AJAX is the name given to a set of modern web application development technologies, previously known as *DynamicHTML (DHTML)* and *remote scripting*, to provide a more interactive web-based user interface.As defined by Garrett [14], AJAX incorporates: standards basedpresentation using XHTML and CSS, dynamic display and interaction using the Document Object Model, data interchange and manipulation, asynchronous data retrieval using XMLHttpRequest, and JavaScript binding everything together. This definition, however, only focuses on the clientside of the web application setting. AJAX is an approach to web application development utilizinga combination of established web technologies. It is the combination of these technologies that makes AJAXunique and powerful on the Web.Even before the term AJAX was coined, its power was becomingevident by web applications such as Google Suggest and Google Map. Other well known examples are Flickr,Gmail, and the new version of Yahoo Mail.

2.2. Frameworks

Web application developers have struggled constantly with the limits of the HTML page-sequence experience, and the complexities of client-side JavaScript programming to add some degree of dynamism to the user interface. Issues regarding cross-browser compatibility are, for instance, known to everyone who has built a real-world web application. The rich user interface (UI) experience AJAX promises comes at the price of facing all such problems. Developers are required to have advanced skills in a variety of Web technologies, if they are to build robust AJAX applications. Also, much effort has to be spent on testing these applications before going in production. This is where frameworks come to the rescue. At least many of them claim to,Because of the momentum AJAX has gained, a vast number of frameworks are being developed. The importance of bringing order to this competitive chaotic world becomes evident when we learn that 'almost one new framework per day is being added to the list of known frameworks. We have studied and experimented with several AJAX frameworks trying to understand their architectural properties.We summarize three of these frameworks in this section.Our selection includes a widely used open source framework called Echo, the web framework offered by Google called GWT, and the commercial package delivered by Backbase.All three frameworks are major players in the AJAX market,and their underlying technologies differ substantially.

Echo2

Echo23 is an open-source AJAX framework which allows the developer to create web applications using an objectoriented,UI component-based, and event-driven paradigm for Web development. Its Java *Application Framework* provides the APIs (UI components, property objects, and event/
listeners) to represent and manage the state of an application and its user interface.All functionality for rendering a component or communicating with the client browser is specifically assembled in aseparate module called the *Web Rendering Engine*. The engine consists of a server-side portion (written in Java/J2EE) and a client-side portion (JavaScript). The client/server interaction protocol is hidden behind this module and as such,it is entirely decoupled from other modules. Echo2 has an *Update Manager* which is responsible for tracking updates to the user interface component model, and for processing input received from the rendering agent and communicating it to the components.The *Echo2 Client Engine* runs in the client browser and provides a remote user interface

to the server-side application. Its main activity is to synchronize client/server state when user operations occur on the interface.A *ClientMessage* in XML format is used to transfer the client state changes to the server by explicitly stating the nature of the change and the corresponding component IDthe change has taken place on. The server processes the ClientMessage, updating the component model to reflect theuser's actions. Events are fired on interested listeners, possibly resulting in further changes to the server-side state of the application. The server responds by rendering a *ServerMessage* which is again an XML message containing directives to perform partial updates to the DOM representation on the client.

GWT

Google has a novel approach to implementing its AJAX framework, Google Web Framework (GWT).Just likeEcho2, GWT facilitates the development of UIs in a fashion similar to AWT or Swing and comes with a library ofwidgets that can be used. The unique character of GWT lies in the way it renders the client-side UI. Instead of keeping the UI components on the server and communicating the state changes, GWT compiles all the Java UI components to JavaScript code (compile-time). Within the components the developer is allowed to use a subset of Java 1.4 API to implement needed functionality.GWT uses a small generic client engine and, using the compiler, all the UI functionality becomes available to the user on the client. This approach decreases round-trips to theserver drastically. The server is only consulted if raw data is needed to populate the client-side UI components. This is carried out by making server calls to defined services. The services (which are not the same asWeb Services) are implemented in Java and data is passed both ways over the network using serialization techniques.

Backbase

Backbase5 is an Amsterdam-based company that provided one of the first commercial AJAX frameworks. The framework is still in continuous development, and in use by numerous customers world wide.A key element of the Backbase framework is the Backbase Presentation Client. This a standards-based engine written in Javascript that runs in the web browser. It can be programmed via a declarative user interface language called BXML. BXML offers library of UI controls, a mechanism for attaching actions to them, as well as facilities for connecting to the server asynchronously. The server side of the Backbase framework is formedby BJS, the Backbase Java Server. It is built on top of JavaServer Faces (JSF)6, the new J2EE presentation architecture. JSF provides a user interface component-based framework following the model-view-controller pattern. The interaction in JSF is, however, based on the classical page sequence model, making integration in a single page framework non trivial.Backbase Java Server provides its own set of UI components and extends the JSF framework to provide a single page interface implementation. Any Java class that offers getters and setters for its properties can be directly assigned to a UI component property. Developers can use the componentsdeclaratively (web-scripting) to build an AJAX application.The framework renders each declared server-sideUI component to a corresponding client-side (BXML) UI component,and keeps track of changes on both component trees for synchronization.The state changes on the client are sent to the server on certain defined events. These can be action events like clicking a button, or value change events such as checking a radio button. The server translates these state changes and identifies the corresponding component(s) in the server component tree. After the required action, the server renders the changes to be responded to the engine again in BXML format.

2.3. Features

While different in many ways, these frameworks share some common architectural characteristics. Generally, the goals of these frameworks can be summarized as follows:

• Hide the complexity of developing AJAX applications -which is a tedious, difficult, and error-prone task,

• Hide the incompatibilities between different web-browsers and platforms,

• Hide the client/server communication complexities,

• All this to achieve rich interactivity and portability for end users, and ease of development for developers.

The frameworks achieve these goals by providing a library of user interface components and a development environment to create reusable custom components. The architectures have a well defined protocol for small interactions among known client/server components. Data needed to be transferred over the network is significantly reduced. This can result in faster response data transfers. Their architecture takes advantage of client side processing resulting in improved user interactivity, smaller number of round-trips, and a reduced web server load.

3. ARCHITECTURAL STYLES

3.1. Terminology

In this paper we use the software architectural concepts and terminology as used by Fieldind which in turn is based on the work of Perry and Wolf . Thus, a software architecture is defined as a configuration of architectural elements—processing, connectors, and data— constrained in their relationships in order to achieve a desired set of architectural properties.An architectural style, in turn, is a coordinated set of architectural constraints that restricts the roles of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style.An architectural style constrains both the design elements and the relationships among them in such a way as to result in software systems with certain desired properties.An architectural system can be composed of multiple styles and a style can be hybrids of other styles. Styles can be seen as reusable common architectural patterns within different system architectures and hence the term *architecturalpattern* is also used to describe the same concept .

3.2. Existing Styles

User interface applications generally make use of popularstyles such as Module/View/Controler to describe large scale architecture and, in more specific cases, styles like C2to rely on asynchronous notification of state changes and request messages between independent components.Many different network-based architectural styles , such as client/server, n-tier , and Code on Demand,exist but in our view the most complete and appropriate style for the Web, thus far, is the REpresentational State Transfer(REST) .REST emphasizes the abstraction of data and services as resources that can be requested by clients using the resource's name and address, specified as a Uniform Resource Locator (URL) . The style inherits characteristics from a number of other styles such as client/server, pipe-and-filter, and distributed objects.The style is a description of the main features of the Web architecture through architectural constraints which have contributed significantly to the success of the Web. It revolves around five fundamental notions: a *resource* which can be anything that has identity, e.g., a document or image, the *representation of a resource* which is in the form of a media type, *synchronous request-response interaction* over HTTP to obtain or modify representations, a *web page* as an instance of the application state, and *engines* (e.g.,browser, crawler) to move from one state to the next.REST specifies a client-stateless-server architecture inwhich a series of proxies, caches, and filters can be used andeach request is independent of the previous ones, inducingthe property of scalability. It also emphasizes on a uniforminterface between components constraining information tobe transferred in a standardized form.

3.3. A Style for Ajax

AJAX applications can be seen as a hybrid of desktop and web applications, inheriting characteristics from bothworlds. Can we reuse styles from these worlds? User interface styles such as C2 are meant specifically forpeer-to-peer environments and thus are not suitable for Web applications.AJAX frameworks provide back-end services through UI components to the client in an event-driven style whereasREST provides resources. AJAX architectures are also not so easily captured in REST, due to the following differences:

• While REST is suited for large-grain hypermedia data transfers, because of its uniforminterface constraint it is not optimal for small data interactions required in AJAX applications.

• REST focuses on a hyper-linked resource-based interaction in which the client requests a specific *resource*.

In contrast, in AJAX applications the user interacts with the system much like in a desktop application, requesting
a response to a specific *action*.

• All interactions for obtaining a resource's representation are performed through a synchronous requestresponse
pair in REST. AJAX applications, however, require a model for asynchronous communication.

• REST explicitly constrains the server to be stateless, i.e. each request from the client must contain all the information necessary for the server to understand the request. While this constraint can improve scalability, the tradeoffs with respect to network performance and user interactivity are of greater importance when designing an AJAX architecture.Because of these requirement mismatches, we do not see how existing styles such as REST or C2 can help to address some of the questions raised in the introduction. Therefore, we will propose a style specifically tailored towards AJAX applications, and study if this style can be used for this purposeinstead.

4. RELATED WORK

While the attention for rich Internet applications in general and AJAX in particular in professional magazines and Internet technology related web sites has been overwhelming, few research papers have been published on the topic so far.Recently a number of technical books have appeared on the subject of developing AJAX applications. Asleson and Schutta focus primarily on the client side aspects of the technology and remain 'pretty agnostic' to the server

side. Crane et al. provide an in-depth presentation of AJAX web programming techniques and prescriptions forbest practices with detailed discussions of relevant design patterns. They also mention improved user experience andreduced network latency by introducing asynchronous interactions as themain features of such applications. While these books focus mainly on the implementation issues, our work examines the architectural design decisions and properties from an abstraction level by focusing on the interactions between the different client/server components.

Pace is an event- based architectural style for trust management in decentralized applications. TIGRA is adistributed system style for integrating front-office systems with middle- and back-office applications. Aura , an architectural framework for user mobility in ubiquitous environments, uses models of user tasks as first class entities to set up, monitor and adapt computing environments. Khare and Taylor evaluate and extend REST for decentralizedsettings and represent an event-based architectural style called ARRESTED. The asynchronous extension of REST, called A+REST, permits a server to broadcast notifications of its state changes to 'watchers'. This work is highly related to the concepts of AJAX applications. Applying a real push-based interaction style to AJAX, however, will probably take some time as the standard browsers and servers do not support this form of communication yet.

The SPIAR style itself draws from many existing styles and software fields, discussed and referenced in the paper. Our work relates closely to the software engineering principles of the REST style . While REST deals with the architecture of the Web as a whole, SPIAR focuses on the specific architectural decisions of AJAX frameworks.

5. CONCLUSION

In this paper we have discussed SPIAR, an architectural style for AJAX. The contributions of this paper are in two research fields: web application development and software architecture. From a software architecture perspective, our contribution consists of the use of concepts and methodologies obtained from software architecture research in the setting of AJAX Internet applications. Our paper

further illustrates how the architectural concepts such as properties, constraints, and different types of architectural elements can help to organize and understand a complex and dynamic field such as single page Internet development. In order to do this, our paper builds upon the foundations offered by the REST style, and offers a further analysis of this style for the purpose of buildingweb applications with rich user interactivity. From a web engineering perspective, our contribution consists of the SPIAR style itself, which captures the guiding software engineering principles that practitioners can use
when constructing and analyzing AJAX frameworks as well as applications. The style is based on an analysis of various of such frameworks, and we have used it to address various design tradeoffs and open issues in AJAX applications. Future work encompasses the use of SPIAR to analyze and influence AJAX developments. One route we foresee is the extension of SPIAR to incorporate additional models for representing, e.g., navigation or UI components, thus making it possible to adopt a model-driven approach to AJAX development. At the time of writing, we are using SPIAR in the context of enriching existing web applicationswith AJAX capabilities.

6.ACKNOWLEDGEMENTS

Partial support was received from SenterNovem, project Single Page Computer Interaction (SPCI). We thank Bas Graaf (TU Delft), Tijs van der Storm (CWI), and Mark Schieffelbein (Backbase) for their feedback on our paper. We particularly would like to thank Kees Broenink (Backbase) for our earlier collaboration on SPIAR .

7.REFERENCES

[1] R. Asleson and N. T. Schutta. *Foundations of Ajax*. Apress, 2005.

[2] D. J. Barrett, L. A. Clarke, P. L. Tarr, and A. E.Wise. A framework for event-based software integration. *ACM Trans. Softw. Eng. Methodol.*, 5(4):378–421, 1996.

[3] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice, 2nd ed.* Addison-Wesley, 2003.

[4] T. Berners-Lee, L. Masinter, and M. McCahill. *RFC 1738: Uniform Resource Locators (URL)*, 1994.

[5] C. Bouras and A. Konidaris. Estimating and eliminating redundant data transfers over the Web: a fragment based approach: Research articles. *Int. J. Commun. Syst.*, 18(2):119–142, 2005.