1-1-2007

Evolving Architectural Patterns For Web Applications

David Parsons
Massey University, d.p.parsons@massey.ac.nz

104. Evolving Architectural Patterns For Web Applications

David Parsons
Massey University, Auckland, New Zealand
d.p.parsons@massey.ac.nz

Abstract
Web application architectural component relationships have evolved over the last decade or so to the point where they have become well established both as common design patterns and embedded in software frameworks. However with the increasing adoption of Web 2.0 technologies and Ajax based web applications, new patterns are starting to emerge. These patterns have yet to become well established in the literature, though a number of new frameworks are beginning to appear. In this paper we review the core patterns of traditional web application architectures, as described in the literature. We then move on to collect some new patterns that have begun to emerge and integrate them into a larger architectural view of how contemporary web applications are evolving. Where it is necessary to illustrate these patterns within a specific web technology, we use components from the Java Enterprise Edition.

Keywords: Web application, design pattern, software architecture, The Web 2.0, Ajax

1. Introduction

Design patterns are reusable elements of software. Their common level of abstraction is a component that solves a general design problem in a particular context, but does not span an entire application or subsystem (Gamma et al, 1995, Buschmann et al, 1996). In contrast, architectural patterns are shared understandings of the major components of a system's design (Fowler, 2003). The relationship between them is that an architecture can incorporate many patterns. For example the JUnit framework has an architecture that utilises a large number of design patterns (Gamma, 1998). In the web application context, the design patterns that are in common use have evolved slowly along with the technologies. For example, the first Java server side component was the servlet in 1996. This was followed by the JavaServer Page (JSP) in 1999. The first JSP specification included two suggested small scale architectural patterns, JSP Model 1 (where a single component processes both the HTTP request and response) and JSP model 2 (where one component processes the request and delegates to another to process the response). Although these were removed from later specifications into supporting documents (e.g. Mahmoud 2003), developers began to build Model 2 architectures with servlets, developing the *front controller servlet* pattern that we see used by popular web application frameworks such as Struts and JavaServer Faces. Thus we find that patterns emerge from practice in ways that are not necessarily foreseen when the technology is first introduced. Currently, we are at a stage where a large number of technologies that have been maturing slowly for some time (JavaScript, the Document Object Model, XML services) are beginning to be used in ways that show emerging patterns and architectures. We begin this paper by reviewing the core patterns that are used in many web application architectures, and describe a reference architecture built on these patterns. We then explore some emerging Web 2.0 and Ajax design patterns that have begun to be described in the literature. We conclude by looking at how these patterns may be integrated into existing architectures to provide an overall architectural pattern for contemporary web application development.

2. Common Web Application Design Patterns

The architectures of web applications and many of their supporting frameworks have become well established since web application components began to appear in the latter half of the 1990s. These architectures are combinations of well understood design patterns that work together to provide architectural frameworks. On the server side, the Model 2 architecture has become widely applied as the view/controller pair of a web-based version of the model view controller architecture, its essential pattern comprising a separation of concerns between the components processing the HTTP request, accessing the underlying model and building the HTTP response (Seshadri, 1999). Behind this view controller layer the model is likely to use patterns such as *data transfer / access object*. The controller itself may be based on either the *page controller* or *front controller* patterns (Fowler 2003). However with frameworks, the latter is much more likely. Another important pattern is the *view helper*, which may to some extent be seen to be in conflict with the front controller as it appears to be based on a Model 1 architecture. However this is only the case if we take the patterns as complete and separate units. Aspects of both patterns can easily be combined if the view helper is seen in a Model 2 context. Further patterns can be applied to the view component itself. A common approach is to use the *template view*, (Fowler, 2003) which is the one expressed by the view helper pattern where the view page contains the template structure for the page and the embedded components (the 'value beans') are plugged into that structure to give dynamic content. However an alternative way of building the view is to use the *transform view* pattern where the page is generated by transforming a (dynamically created) XML document. The two may be combined by plugging partial transforms that generate parts of a document, rather than a full page, into a template view. Figure 1 shows how these various patterns fit into a web application architecture. Although this blending of architectural and design patterns is common in practice, we rarely find them combined together in this type of representation. It is important, however, to begin our analysis with this kind of overview because each interaction between an architecture and a pattern, or between one pattern and another, has implications for the functionality and cohesion of the overall application. Selecting between a template or transform view, for example, has major implications for the way the system works even though both can be encompassed within a Model 2 architecture.

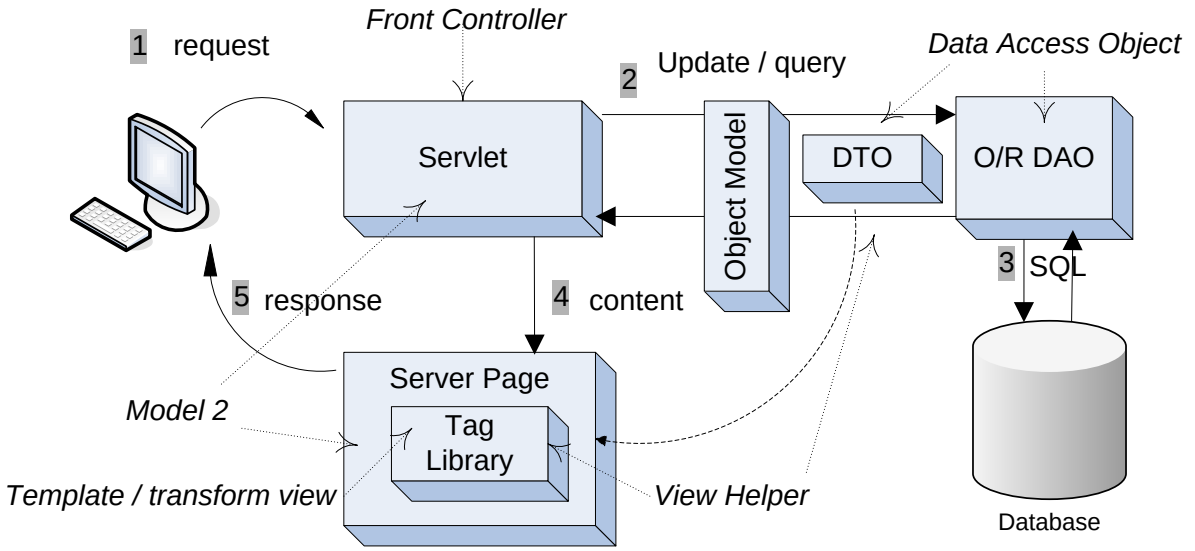

Figure 1: Common design patterns within a web application architecture

On the client side, patterns are less well defined. For a totally thin client, which only renders markup, the only applicable patterns are those that relate to page sequencing, for example 'wizard' workflows where a series of form pages are sequenced one after the other. Many of the other published patterns that relate to thin client architectures focus mostly on usability patterns (e.g. Graham 2003) rather than software architectures.

3. The Emergence of the Web 2.0 and Ajax

The established web application patterns and architectures discussed in the previous section are still widely used. However the recent emergence of the Web 2.0 and Ajax (Asynchronous JavaScript and XML), which Booch (2006) suggests marks the move between 5th generation and 6th generation web applications, have led to further developments in the architectures of web applications to encompass these new approaches to web development, and new patterns are beginning to emerge as a result. Rich Internet Application Frameworks are also beginning to appear (Shan and Hua, 2006).

3.1 The Web 2.0

The Web 2.0 is a term that has become widely used since the first Web 2.0 Conference in 2004. Although it might be categorized as an umbrella marketing term rather than a specific technology or architecture, some authors, notably O'Reilly (2005), have given it some concrete specifications through a set of published principles, practices and patterns. Many publications that discuss the Web 2.0 focus on rich user interfaces, in particular the use of Ajax, but the ideas of the Web 2.0 go beyond Ajax to include a wide range of ideas about how modern web applications should be developed. The key ideas underlying the Web 2.0 may perhaps be summarized as:

- The web as a software platform
- Service oriented architectures
- User and contributor communities

3.2 The web as a software platform

In the past, the software platform that applications were built on was a particular computer operating system, for example Microsoft Windows or Linux. In contrast Web applications are able to span multiple operating systems because web browsers can render the same content regardless of the original system from which the page was downloaded. Further, 'smart' clients can be integrated into applications that run over the web. For example, to download music we might use a PC to connect to a web server and also connect a mobile device to the PC, all using a single application. In this type of situation, the software platform that the overall application is running on is the Web, not just a single device.

3.3 Service oriented architectures

In the early days of the Web, the focus was on the applications that were being used. For example the 'browser wars', primarily between Netscape and Microsoft in the mid 1990s, were about which application would be used to access the Web. More recently, the focus has been more on the underlying content available via the Web, rather than the specific applications that might be used. This content is made available using various forms of web service, which are data sources made available over the web using the eXtensible Markup Language (XML). One simple example of a web service is RSS (an acronym that has multiple roots, Really Simple Syndication, Rich Site Summary and RDF Site Summary), which uses XML to supply feeds of frequently updated information such as news and weather.

3.4 User and contributor communities

Traditional software construction is about building self-contained applications for a particular purpose. In many Web 2.0 applications, instead of this type of central control, applications are about a community of users who participate in the application itself. A good example of this is Wikipedia, the on-line encyclopedia where anyone can create or edit entries. Of course opening up a web application to contributions from the user community is not appropriate for every system, but certain aspects of the approach to software development can be incorporated into many different types of web application.

3.4 Ajax

Asynchronous JavaScript and XML (Ajax) is a term coined by Garret (2005). At its simplest, Ajax makes it possible to update parts of a web page with data read from a server without having to refresh the whole page, making the user experience more like using a traditional desktop application rather than surfing a web site. Ajax itself is not a technology but rather a grouping of complementary technologies. Garret summarized Ajax as a combination of:

- Standards based presentation using XHTML and CSS
- Dynamic display and interaction using the Document Object Model (DOM)
- Data interchange and manipulation using XML and eXtensible Stylesheet Language Transformations (XSLT)
- Asynchronous data retrieval using the XMLHttpRequest
- JavaScript binding everything together

Figure 2 shows the general architecture of Ajax based systems. The key to this architecture is that the Ajax engine mediates between the user interface and the server, processing on the client where possible (using Dynamic HTML) and, where necessary, sending asynchronous HTTP requests and receiving XML data (or indeed data in any other suitable format) that it renders in the browser via the DOM.

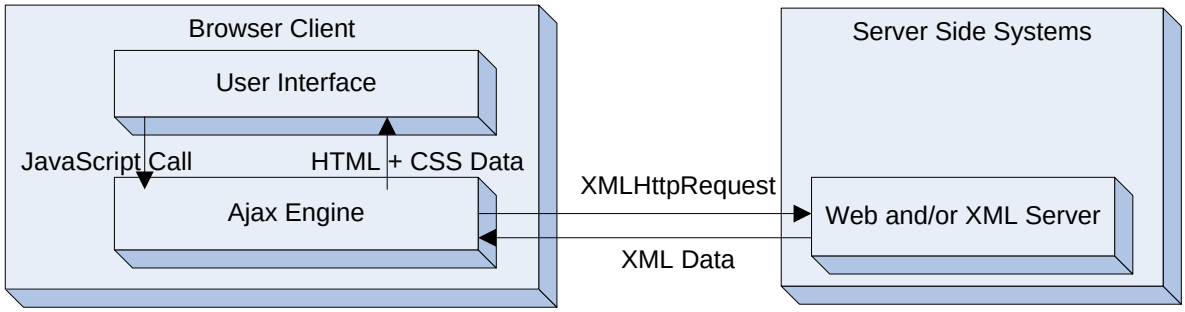

Figure 2: Ajax architecture (adapted from Garrett (2005))

Equally importantly this processing can take place asynchronously. This means that the user does not have to wait for the server to respond in order to continue interacting with the application. Instead, the application is able to continue serving the user while at the same time handling the server response as and when it arrives. User activity in the browser continues even while the Ajax engine is submitting XMLHttpRequests to the server and waiting for responses. The Ajax engine is responsible for handling events associated with getting back the server response but the user does not have to wait for it. Ajax applications do not have to be asynchronous, however. In some cases it might be appropriate to wait for the server's response before continuing with the current process.

Two major benefits that have been claimed for Ajax applications are improvements in both system and user performance, supported by a reduction in the interruptions of page reloading and a reduction in network traffic, while the major penalty appear to be in system complexity (Paulson 2005). Although published research is so far limited, there have been some attempts to measure Ajax performance, notably Smullen and Smullen (2006), indicating that the reduction in required bandwidth can indeed improve performance measured in terms of browser response times. Research on user performance has not so far been published, though Atterer and Schmidt (2007) discuss a tool that would support this type of analysis. In this paper we focus on the possibility of managing architectural complexity by understanding and integrating some emerging Ajax design patterns.

4 Mining New Web Application Patterns and Architectures

Since the emergence of the Web 2.0 and Ajax, new patterns and frameworks have emerged that challenge aspects of the architectures that we have grown familiar with. However due to the fragmentary nature of Ajax, which is an architectural idea rather than technology, and the many other aspects of the Web 2.0, there is little concrete information with which to identify the underlying patterns. For example, O'Reilly's Web 2.0 patterns (O'Reilly, 2005) are in fact no more then heuristics at best, marketing slogans at worst. At no point do they include any architectural guidelines. Garret's (2005) original Ajax article provided us with some more guidance about Ajax architecture, but did not attempt to specify patterns. Underlying these two aspects is a 'bifurcation' between the move towards the rich client technologies on the one hand, and service oriented architectures on the other (Booch 2006). The conflict between these two may not always be obvious, but can lead to some interesting problems, For example, it is problematic to build an Ajax client (rich client) for an RSS feed (service) because of the security restrictions of browsers. At best, you have to negotiate a warning message, at worst, the browser will refuse to connect to a third party server.

Fortunately some pattern based work is beginning to emerge that can help us to identify some useful architectural components that may be reused across different web application implementations. To some extent, Ajax has exhibited the characteristics of what Booch refers to as 'accidental architecture'. Importantly, he says that "by naming these accidental architectures, we again raise the level of abstraction by which we can describe and reason about a system" (Booch 2006, p.10). At present, we are beginning to see a number of Ajax patterns emerge but few architectural aspects. The Wiki based Ajax patterns page, for example, includes (at the time of writing) around 60 small scale pattern descriptions, but no patterns under the heading of 'Ajax Architecture' (Mahemoff 2007). In the following section we make some initial efforts to mine the literature for some patterns that could be regarded as being applicable at the architectural level.

4.1 Auto-completion architecture

MacLanahan (2006) provides a sequence diagram representation of a conceptual implementation of auto completion, one of the most evident features of an Ajax implementation. In this pattern the JavaScript component creates and initializes the XMLHttpRequest object which connects to a servlet. The response is, of course, indicated as asynchronous and the final result is to update the Document Object Model (DOM). He further describes the server side implementation of an autocomplete process on the server using JavaServer Faces as an example framework. Though this is a framework specific example, the underlying patterns can be generalized to any Ajax application. There is a server side listener applied to each component, and further server side components that relate specifically to the client side field that is being used for data entry.

4.2 Delta management architecture

Mesbah and van Deusen (2006) provide a useful description of the level of data interaction between client and server in an Ajax implementation, as part of their SPIAR architectural style. They define the 'delta encoder / decoder' component on the server, that is responsible for identifying the delta between the previous and current state of the client, and ensuring that the minimum amount of data transfer takes place. This is an important feature of Ajax implementations, since one of the key points about Ajax is that we do not need to refresh an entire page, much of which is duplicated, when all that has changed is one small part of the content.

4.3 Client side buffering

Mahemoff (2007) lists a series of Ajax patterns that he describes as 'architectural', though they are really small architectural components. These include *local event-handling*, which can be supported by a *local cache*, and *predictive download*. To further improve the client server efficiency, he proposes various alternative strategies such as *submission throttling* or *explicit submission*. All of these architectural concepts can be merged at a higher level of abstraction into a component of the Ajax engine that manages interaction by ensuring that the number of XMLHttpRequests made to the server is minimized. We will refer to this general component as the XMLHttpRequest buffer.

Figure 3 shows how the three general architectural Ajax patterns might be integrated into a Web application architecture. The XMLHttpRequest Buffer pattern becomes integrated with the Ajax engine on the client to minimize data transfer to the server. On the server side, the auto completion components are chained before the delta encoder/decoder in order to minimize data transfer back to the client.

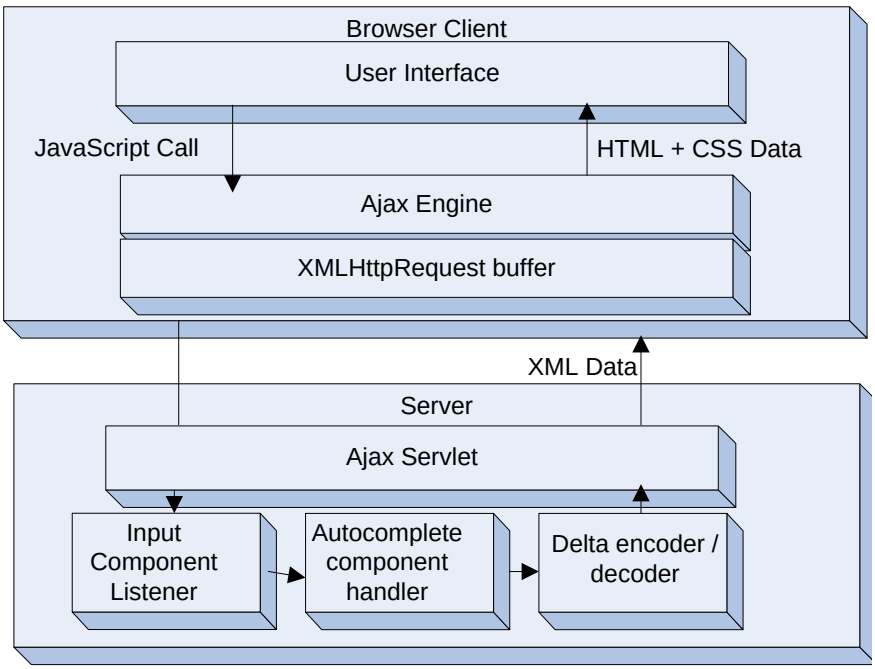

Figure 3: Ajax architectural patterns

5. Summary and future work

This paper describes an ongoing effort to identify emerging architectural patterns in the context of contemporary Web 2.0 applications using Ajax. As with all pattern mining activity the intention is to identify best practice from the work of others and re-present it in such a way that it can be successfully reused. Given the current stage of development of Ajax tools and frameworks, there is much more work to be done in this area, but we have identified a number of useful architectural patterns and outlined how they might be integrated into an overall reference architecture for web application development. However there is much more work to be done in mining a rich set of patterns that can provide a more complete guide to the web software architect that may help to address the complexity inherent in an Ajax-based approach whilst still delivering the demonstrable performance benefits.

References

Atterer, R. and Schmidt, A. "Tracking the Interaction of Users with AJAX Applications for Usability Testing," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, San Jose, USA, 2007, pp. 1347-1350.

Booch, G. "The Accidental Architecture," *IEEE Software* (23:3), May/June 2006, pp. 9-11.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley, Chichester, 1996.

Fowler, M. *Patterns of Enterprise Application Architecture*, Addison-Wesley, Boston, 2003.

Gamma, E. "Junit: a Cook's Tour", Retrieved January, 2007, from http://junit.sourceforge.net/doc/cookstour/cookstour.htm, 1998

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass., 1995.

Garrett, J. "Ajax: A New Approach to Web Applications," Retrieved January, 2007, from http://www.adaptivepath.com/publications/essays/archives/000385.php, 2005.

Graham, I. *A Pattern Language for Web Usability*, Addison-Wesley, London, 2003.

McClanahan, C. "The State of Web Frameworks," Retrieved March, 2007 from. http://kr.sun.com/developers/PDFs/preso/Craig_JCO2006.pdf , 2006

Mahemoff, M. "Ajax Patterns: Design Patterns for Ajax Usability," Retrieved March 2007 from http://softwareas.com/ajax-patterns, 2007

Mahmoud, Q. "Servlets and JSP Pages Best Practices," *Sun Developer Network*, 2003, Retrieved January, 2007 from http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/

Mesbah A. and van Deursen, A. "An Architectural Style for Ajax'", *Delft University Software Engineering Research Group Technical Paper*, 2006.

O'Reilly, T. "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software," *O'Reilly Network*, Retrieved January, 2007, from http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, 2005.

Paulson, L. "Building Rich Web Applications with Ajax," *IEEE Computer* (38:10), 2005, pp.14-17.

Seshadri, G. "Understanding JavaServer Pages Model 2 Architecture: Exploring the MVC Design Pattern," *JavaWorld*, December 1999.

Shan, T. and Hua, W. "Taxonomy of Java Web Application Frameworks," *Proceedings of IEEE International Conference on e-Business Engineering (ICBE'06)*, Shanghai, China, 2006, pp.378-385.

Smullen C. and Smullen S. "Modelling AJAX Application Performance," *Proceedings of Web Technologies Applications and Services*, J.Yao (ed.), Calgary, Canada, 2006.