

An Analysis of Approaches for Asynchronous Communication in Web Applications

Stefan Potthast, M.Sc.
Computer Science Department
University of Applied Sciences
Darmstadt, Germany
stefan.potthast@sopheon.de

Mike Rowe, Ph.D.
Computer Science and
Software Engineering Department
University of Wisconsin
Platteville, WI 53818
rowemi@uwplatt.edu

Abstract

Since the end of 2005 there has been a lively discussion in the Internet Community about the movement to a new paradigm called Web 2.0. Part of this revolution is the concept of asynchronous communications. This paper studies four different techniques/technologies for achieving asynchronous web communications. The study includes the analysis of implementation issues, features, drawbacks, performance and complexity measurements of approaches used to achieve user experienced asynchronous communication in web applications. The AJAX approach was easy to implement and well supported in browsers and frameworks. Other approaches were inferior to AJAX. HTML Inline Frames are difficult to handle, because they were not originally designed to support asynchronous communication. Microsoft RDS were too specialized and also are being replaced by newer technologies. The DOM Level 3 Load&Save is still in a fledgling stage. Because of the intrinsic capabilities of AJAX for implementing asynchronous communication without the constraints and shortcoming of the other technologies was clearly the best choice with respect to both performance on all criteria studied. In the author's opinion, asynchronous communication implemented using AJAX fuels the expansion of Web 2.0, whose evolution will proceed and so AJAX will likely gain more and more importance.

Note: This paper is a short version of the author's master thesis. For accessing the full version contact Stefan Potthast.

1 Introduction

Since the end of 2005 there has been a lively discussion in the Internet Community about the movement to a new paradigm called Web 2.0 and its moving spirit AJAX. A part of Web 2.0 is the implementation of business models into highly interactive web applications, which are designed to close the previous gap between desktop and web applications with respect to usability and latencies. In this context, AJAX stands for Asynchronous JavaScript and XML, and is a combination of existing technologies used to establish user experienced asynchronous communication in web applications. Data can be retrieved and included into currently displayed web pages via asynchronous requests without going the traditional way by reloading the whole web page.

Even though AJAX is strongly connected to the new way of creating Web 2.0 applications there are other technologies or rather techniques already available to perform such asynchronous communication. Known under the name of "Remote Scripting" the mature HTML Inline Frames or the Microsoft Remote Data Services (RDS) were used to achieve comparable functionality. The same applies for fairly new approaches such as DOM(Document Object Model) Level 3 Load and Save.

Although AJAX attracted a great deal of attention in the internet, a comprehensive comparison to the other technologies had not been drawn yet. It should be mentioned admittedly that the founder of the term AJAX, Jesse James Garrett from Adaptive Path, built the fundamental ideas of what is elaborated in this paper in his first article about AJAX – "AJAX: A New Approach to Web Applications" [1].

Noda and Helwig from the University of Wisconsin did a technical comparison and case study of AJAX, Flash, and Java based Rich Internet Applications [2]. The task of the comparison and case study is similar in this paper with the main difference being the technologies. By contrast the focus in this paper is on approaches which implement asynchronous communication without the need of additional software running in the browser. Thus, Java and Flash as seen in [2] are not of interest for this elaboration.

2 Asynchronous Communication

Most of the existing web applications act the traditional way. For example the click on a link or a button initiates the sending or loading of data by triggering a HTTP Request to a web server. After processing the request the web server sends its response, which in turn is presented by the browser. In most cases the response is a new HTML page.

The fact that the synchronous HTTP Request is initiated directly by the User Interface (UI) is crucial, because the UI is blocked until the server responds. But from a UI-Oriented view there are other methods that can trigger HTTP Transactions. Those meth-

ods might have an enormous effect on the experience of the user, regarding for example blocking of a web page during a transaction. Even if the underlying HTTP communication is, due to the HTTP characteristics, always a synchronous communication, a user might get the impression of an asynchronous communication.

A different approach to the direct initiation of HTTP Requests by the UI is an encapsulation of the communication in a separate Transaction and Embedding (T&E) Component.

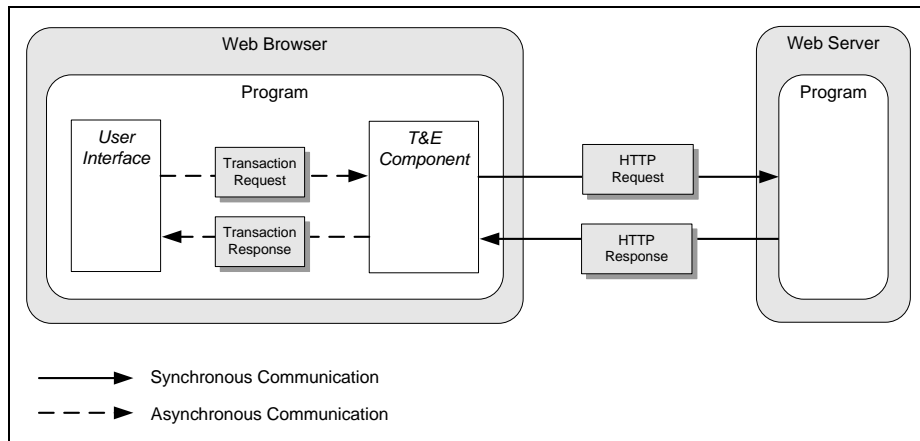


Figure 1: Asynchronous Web Communication Architecture

The Asynchronous Web Communication Architecture shows that synchronous HTTP Requests to a server are not executed directly by the UI. Instead, the T&E Component is used to send and receive data.

In applications based on architectures as shown in Figure 1, the asynchronous behavior becomes apparent by looking at the communication flow. The T&E Component communicates synchronously with the server. However, the UI communicates asynchronously with the T&E Component. This means that the UI is not blocked while waiting for a response after initiating a transaction. Thus, the user is still able to interact with the interface.

Besides the non-blocked characteristic, the UI's asynchronous communication has one other main advantage, that being it reduces latencies. In the traditional synchronous communication, a request to the server for an updated version of the currently displayed web page is answered with a complete version of the HTML page containing the updated information. The bulk of the source code of this result web page is nearly the same as of the previous web page, which initiated the request. In the asynchronous approaches only the new data is actually transferred. The embedding of such data into a currently displayed web page can be implemented by manipulating elements of the DOM-Tree.

3 AJAX

Beginning 2005 Jesse James Garrett's article "AJAX: A New Approach to Web Applications" [1] described a combination of already existing technologies with asynchronous JavaScript and XML (AJAX) using the XMLHttpRequest (XHR) object to realize web applications with a specific behavior. Although actually web applications implementing these techniques already existed at that time, the introduction of the term AJAX made it possible to address them in a clear and brief way. Garrett's article and applications such as Google Maps [3], which made the utility of an AJAX appliance obvious, can be considered catalyst for the hype that followed.

From a technical point of view the XHR object used in the context of JavaScript has focus, because it provides the possibility to do the asynchronous processing. The XHR interface is not standardized as many other objects or rather languages that are defined for instance by W3C specifications. Instead it was introduced by Microsoft some years ago as an ActiveX component. Over the years, Microsoft evolved the interface and the other browser makers added comparable components to their software with similar names and nearly the same interfaces. A main difference to Microsoft's solution is that the other browser makers implemented the XHR object natively into their browser so no additional software support like ActiveX is needed to use it. In the newest version of the Internet Explorer, version 7, this object is implemented natively, too. Nevertheless, due to compatibility issues, if using the XHR object nowadays, developers have to write different object creation code in cross-browser implementations.

```
01  function startAsynchCall(intDataId)
02  {
03      var objXmlHttp = CreateXmlHttpRequestObject();
04      objXmlHttp.open('POST', 'getData.asp', true);
05      objXmlHttp.setRequestHeader('Content-Type',
06          'application/x-www-form-urlencoded; charset=UTF-8;');
07      objXmlHttp.onreadystatechange = asynchCallHandler;
08      objXmlHttp.send('id=' + escape(intDataId));
09  }
10  function asynchCallHandler()
11  {
12      if (objXmlHttp.readyState == 4)
13      {
14          if(objXmlHttp.status == 200)
15          {
16              includeDataToPage(objXmlHttp.responseText);
17          }
18          else
19          {
20              alert("Failure while retrieving information!");
21          }
22      }
23  }
24  function includeDataToPage(strData)
25  {
26      //DOM tree manipulation is done here.
27  }
28  }
```

Code Fragment 1: AJAX – XHR Usage

XHR objects can be used when a web page is requesting data from a server by triggering a JavaScript function *startAsynchCall* as shown in Code Fragment 1. Once the XHR ob-

ject is created in this function, its *send* method can be used to start a communication with the server. The script performed at the server-side has to be defined prior in the *open* method, where also the HTTP method and synchronism mode are set. The function *asynchCallHandler* is assigned to the *onReadyStateChange* event as handler for every state change of the XHR object. Following the call of the *send* method for an XHR object in asynchronous mode, two things happen. First, the *startAsynchCall* function terminates. Second, the still existing XHR object runs through all five possible states from 0 to 4 namely: uninitialized, loading, loaded, interactive and complete. While the object is waiting to reach the next state other operations or rather function can be performed. By the time a new state is reached the handler is called. In Code Fragment 1 *asynchCallHandler* includes the received data into the currently display web page in case of a complete and successful transaction.

Thus, a T&E Component built with AJAX technology could consist of the three introduced functions. It is obvious that the implementation is not very complex at all. Especially because in web applications consisting of HTML, ASP, JavaScript, CSS, XML, XSLT or other files the consolidataion of all needed functionality together within one JavaScript file is of value. That is because the more the implementation of certain functionality is spread across the application files, the more difficult it gets to follow the process. In contrast to that, the encapsulation of functionality into only one JavaScript function makes it much easier to debug, maintain, or extend code. Here, the Code Dispersion is a metric defined by the author of this paper, describes the spread of functionality-bound source code across the whole web application.

To set up a value for the Code Dispersion, first the main block of the functionality has to be identified. In most cases it will be a function or a set of functions, where the functionality is being implemented. The Dispersion, *D*, is the ratio between lines of code in the main block, LOC_i , and lines of code outside the main block, LOC_o , which are necessary to execute the implemented functionality. Outside lines are, for example, an object inclusion in the main part of a web page, or an HTML form. The correspondent equations are as follows.

$$LOC = LOC_i + LOC_o \qquad D = \frac{LOC_o}{LOC}$$

Figure 2: Definition of the Code Dispersion Complexity Measurement Metric

In the opinion of the author the low complexity of AJAX, especially with respect to its Code Dispersion value of 0 % measured in test implementations done for this paper, is a feature of this approach.

Based on the fact that reduction of latencies is one of the main goals of asynchronous communication in web application the performance of AJAX has been measured and compared to the other approaches. Maybe the most apparent mark for the AJAX approach is the missing of two measures for performing a saving of test data. Opera was not able to save data having a size of more than approximately 5 kb sent with the HTTP

POST method. This is neither a restriction of HTTP nor of the objects that have been used. Especially a comparable behavior with the Inline Frame technique, which will be introduced in the next section, showed that it is likely a browser limitation. However, even in the developers interface for advanced browser settings¹ for Opera no configuration of that limit was possible.

The second particularity might not be as apparent as the missing of two measures but is still distinctive. That is the loading times with Firefox were not only significantly more or less higher than with other browsers, but also nearly equal for data of 1 and 10 kb size. This behavior appeared reproducible with different configurations and also with Internet Explorer. There seems to be a lower bound of data amount below which the browser shows no difference concerning the time needed to perform a loading. Further tests with only 1 Byte data produced the same loading time and have shown that the mentioned lower bound is approximately 5 kb. The only significant difference compared to the other browsers occurred when dealing with small data amounts as Figure 2 shows.

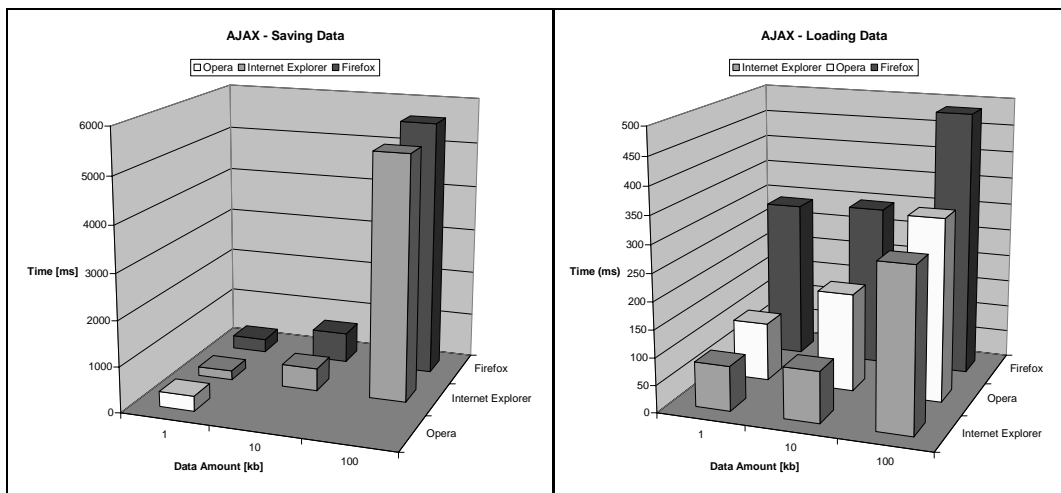


Figure 2 - AJAX Saving and Loading Times Histograms (Truncated Means)

In addition to its low complexity, the features of AJAX are its large number of available free frameworks for example Prototype [4] or the Dojo Toolkit [5] and the already emerged patterns and best practices described amongst others by Christian Gross [6]. All this features help to ease development of high quality web applications.

Drawbacks as unavailability of browser functionalities such as Bookmarking or Back/Forward-Buttons, both related to breaking the page metaphor of the web, belong to asynchronous communication and DHTML in common instead of being a typical AJAX problem.

4 Other Approaches

¹ An interface to the complete for users available settings of Opera can be accessed via typing “opera:config” in the address line.

4.1 HTML Inline Frames

This approach is one of the first applied alternatives to achieve asynchronous communication between web browsers and web servers. Inline Frames, originally introduced by Microsoft, are a special type of frames, which have been part of the HTML Specification [7] since version 4.01. Even though there are several competing approaches and AJAX seems to supersede the Inline Frame technique, at the moment it is still widely used in web development.

The common intention of HTML Inline Frames is to embed resources into a block of HTML documents. Figure 4 sketches the Page-In-Page implementation using an inline frame.

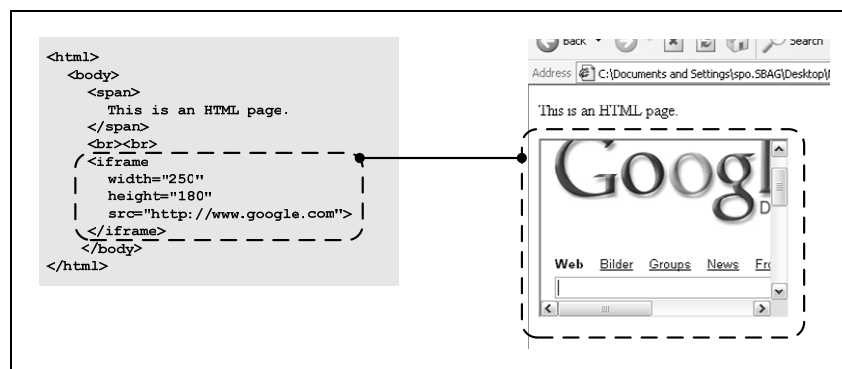


Figure 4 - Page-In-Page Using Inline Frames

The basic idea is to use such inline frame elements to retrieve data in terms of a web page inside of the element. This is a standard synchronous HTTP transaction, which does not affect the performance of the surrounding HTML page. Basically, an inline frame is an independent web page within another web page. By the time a new web page is completely loaded within the inline frame the “main” HTML page can access the content of the inline frame and process it using DHTML methods on the DOM tree. Often inline frame elements are hidden, so that the user does not notice any part of the loading procedure. A further introduction to HTML Inline Frames in the remote scripting context can be found in the Apple Developer Connection Portal [8].

A main advantage of the HTML Inline Frame approach is the cross-browser compatibility. The *iframe* element itself is supported by all used browsers (Internet Explorer, Firefox, Opera). The test implementations average Code Dispersion was comparably low with a value about 10 %, whereas the performance analysis has shown that problems arise when loading data in Internet Explorer. In other approaches and with HTML Inline Frames in other browsers, the data embedding times were irrelevant evanescent small. However, with Internet Explorer the data embedding times increase exponentially with the amount of data loaded, which might be critical on larger amounts. Also, problems arise with the handling of HTML Inline Frames to track points in time when the loaded data is available. Another drawback is related to browser caching, where specifically XML files produced problems differing from browser to browser, as the tests revealed.

4.2 Microsoft Remote Data Services

Remote Data Services (RDS) is a Microsoft technology. RDS is based on ADO, which is the ActiveX Data Objects. ADO provides some features as connections to databases or containers to carry tabular data. The latter are so called ADO Recordsets, which are used in RDS, too. Namely RDS provides the ability to remotely retrieve Recordsets, cache them on the client-side, and send back changes resulted from manipulations. When RDS is used in web application it is restricted to Internet Explorer, which is the only browser that supports RDS objects.

The bonus with RDS is that developers are able to link the mentioned Recordsets directly to HTML elements. This means with RDS data from a database table can be mapped to an HTML table just by telling the table which field from the database table should be mapped to which column in the HTML table. RDS takes care of putting all data in the table, so no further scripting is needed.

The DataControl is one of the RDS objects. It has to be included to an HTML page and initialized on loading of the page by calling a JavaScript function. JavaScript handler functions like shown in the AJAX chapter are not needed for RDS because the data binding is done automatically. Indeed, in addition to the DataControl object HTML span elements with specific attributes are needed for that data embedding. Unfortunately using RDS as described the asynchronous sending of data is not possible. In general, asynchronous saving of data can be achieved with RDS using the objects DataSpace and DataFactory directly instead of via a DataControl, but their flexibility with respect to connecting different types of data sources disqualified them from being discussed in this paper. Further information and a good introduction to RDS are given in John Papa's book about RDS [9].

With respect to complexity RDS has a high Code Dispersion with over 30 % in contrast to the good performance with very fast loading times.

Currently RDS is very rarely used. Some reasons lay in the technology functionality itself but the biggest issue is that Microsoft has deprecated RDS in the Microsoft Data Access Components (MDAC) [10], a superior Data Access Architecture. The reason is that Microsoft's SOAP Toolkit 2.0 should replace RDS. However the SOAP Toolkit cannot be recommended for used, not only because of the absence of asynchronous communication functionality, but also because in the meantime it is also deprecated. The technology that replaces all the above mentioned is .NET.

4.3 DOM Level 3 Load & Save

DOM Level 3 Load and Save, from now on DOM3 LS, was introduced by the W3C in 2002 with a Working Draft status. Its aim was to extend the DOM with a comfortable

XML document loading and saving functionality. In 2004 DOM3 LS reached the Recommendation status of the W3C and in the abstract of the specification it was described as follows:

“... a platform- and language-neutral interface that allows programs and scripts to dynamically load the content of an XML document into a DOM document and serialize a DOM document into an XML document ... It also allows filtering of content at load time and at serialization time.” [11]

Until mid 2006 only Opera supported some of the DOM3 LS interface² at all. Unfortunately, Opera does not provide very specific information about the support in its Web Specifications Supported website for version 9 of the browser [12]. Therefore, it is up to the developer to find out what works and what does not. The *LSParser* object used for loading and embedding of data works in Opera as described in the specification. Unfortunately, this is not applied to the *LSSerializer* objects for saving data. So only a theoretical inspection was possible. However, even on closer inspection a decisive difference appeared in the comparison of the both objects. As defined in the specification of DOM3 LS, the *LSSerializer* object has no property for setting a synchronism mode as the *LSParser* does. In all likelihood, methods performed on the *LSSerialize* objects will be performed synchronously specially because there is no possibility to add a “save”, DOM3 LS specific, event listener to the object.

The loading tests that were possible run produced result times nearly as fast as with RDS, but DOM3 LS also had low Code Dispersion with 0 %. But, definitely the biggest feature of DOM3 LS is the very potent and useful API. It makes development of asynchronous communication and embedding with respect to data very simple. The methods *parseURI* (*LSParser*) and *writeToURI* (*LSSerializer*) of DOM3 LS uses the lower level objects *LSInput* and *LSOutput*, which represent input or output sources. With these objects, not only can XML sources be represented, but also character streams, byte streams, and string data without XML declarations. Even though DOM3 LS is optimized for XML data, the broader scope of supported data sources is a useful feature.

Further, valuable features of DOM3 LS are the possibility to bind an XML schema validation to the parsing process of XML data and the determination of loading and parsing progresses through events. Both features are not supported by the other approaches to the same extent. DOM3 LS also has strengths in error handling by providing *LSExceptions*.

The *LSParser* object has two not yet mentioned features that are considered very powerful. One is that developers can add a self-defined filter to the parser. This filter can be used to decide which nodes of the parsed XML should be inserted into the document or when the parsing of the XML should be stopped. It is even possible to modify the parsed data through the filter function. The second powerful feature is the method *parseWithContent* with its parameters *input*, *contextArg* and *action*. With this method, data within the input object can be hooked directly into the DOM-Tree at a specific node selected by *contextArg*. Using the parameter *action* the position can be chosen such that the data will

² Apple Safari browsers shall support the interface, too, but they are not discussed in this paper.

be hooked in with respect to the selected node of the DOM-Tree. Possible values for *action* are *APPEND_CHILDREN*, *REPLACE_CHILDREN*, *INSERT_BEFORE*, *INSERT_AFTER*, and *REPLACE*. In other approaches the tasks of loading, parsing and inserting of data have to be implemented and performed separately; in DOM3 LS developers can do all of these together with only one function call. Thus, there are savings not only in writing code, but also in debugging and maintaining it. Finally, less code has to be downloaded to the client, which decreases latency.

Another advantage of DOM3 LS is that it is an API standardized by the W3C. In the long run particularly cross-browser and cross-programming projects will benefit by using DOM3 LS.

5 Comparison

5.1 Functionality

In the AJAX approach the available functionality of the different implementations of the XHR object worked as expected. Thereby all the XHR interfaces reflect cohesive sets of core functionality for asynchronous communication. The process of loading and saving data can be widely customized by using specific HTTP methods and headers. With HTML Inline Frames, developers are limited to the GET and POST operations for the saving task, because the used HTML form provides this definition. The direct loading of Inline Frames by setting the `src` Attribute has no choice of methods at all. In both tasks on HTML Inline Frames, the access to HTTP headers is not possible. RDS and DOM3 LS only provide the possibility to switch between synchronous and asynchronous communication.

Meaningful features such as status information, data access, or abortion of a transaction are not available natively with HTML Inline Frames, but have to be implemented by the developers themselves. This makes XHR objects, RDS and DOM3 LS more convenient to use. Specially DOM3 LS covers the core functionalities and in addition offers broad capabilities in processing and including of data into the DOM-Tree. RDS on the other hand has features as paging, sorting or filtering, but comes with some problems with its core functionalities. Not one of the documented events of RDS fired in testing the approach and the saving of data is limited to specific implementations, which are inappropriate for the tasks in this paper.

5.2 Complexity

The Tables 1 and Table 2 show program length and Code Dispersion of the test implementations for the different approaches.

Approach	Program Length (LOC)	Code Dispersion (D)
AJAX(XHR)	26	0
HTML Inline Frames	29	0.07
Microsoft RDS	13	0.31
DOM3 Load and Save	14	0

Table 1 - Complexity Measures for the Loading Task

Approach	Program Length (LOC)	Code Dispersion (D)
AJAX(XHR)	27	0
HTML Inline Frames	37	0.14
Microsoft RDS	N/A	N/A
DOM3 Load and Save	N/A	N/A

Table 2 - Complexity Measures for the Saving Task

Concerning program length there are significant differences between the covered approaches. With Microsoft RDS and DOM3 LS only a small amount of code is needed to implement functionality in contrast to AJAX and especially HTML Inline Frames, where code size is nearly doubled. Mainly AJAX outrivals HTML Inline Frames in less complexity, because the Code Dispersion of the latter is larger. For the RDS approach, the Code Dispersion value is comparatively high, but related to the short program length this negative point is lessened. DOM3 LS came in with outstanding performance and because of the little code that is needed, there is no Code Dispersion in the implementation.

5.3 Maintainability

An all-time problematic issue in maintainability of web applications has been debugging. The standard way, to interrupt the processing by printing debug information directly to the display, nowadays is accompanied by some new features of the browsers. For example, Opera and Firefox have integrated consoles to track JavaScript errors. Assistance, such as modern development environments provide it to programming languages as C# or Java like debugging environments or code completion, is still not common for web development. With asynchronous communications it gets worse because things are performed in parallel and in background, what makes it important, is that the approaches support error identification and error handling. DOM3 LS supports debugging and handling of run-time errors by providing specific error types and exceptions. Concerning documentation, RDS also provides errors and error events, but due to the fact that these events are not fired correctly they are currently of no value. The XHR objects of AJAX as well as HTML Inline Frames do not have anything comparable in this scope.

Another point of maintainability is how extensible the program or rather functionality is depending on the selected approach. With its object-oriented capabilities and a good application design, JavaScript makes it fairly simple to extend existing or add new func-

tionalities. Problems may result if parts of the implementation of functionality are located outside of the JavaScript blocks, for example including *<object>* tags or custom attributes to HTML elements. The results of the complexity measurements have shown that the Code Dispersion of HTML Inline Frames as well as RDS is relatively high. This definitely makes it more difficult to extend functionality, starting with the search for the right file and position in it. Not only extensions, but also debugging and error corrections are more difficult if Code Dispersion is high.

5.4 Browser Compatibility

The most browser compatible approach nowadays is HTML Inline Frames. It uses only standard compliant HTML elements and DOM manipulations, thus no special browser settings were needed in the used browsers.

The support of AJAX is comparably good, as every browser provides a capability to instantiate an XHR object. Opera and Firefox already have a native implementation, whereas Internet Explorer currently uses an ActiveX component but is expected to supply a native implementation in version 7. ActiveX capabilities must be explicitly enabled in Internet Explorer, otherwise the XHR object, and thereby AJAX, cannot be used. The security issues arising by enabling ActiveX in current versions of Internet Explorer should have a negative impact in the assessment of AJAX browser compatibility. However, as version 7 of Internet Explorer will implement the XHR interface natively, this devaluation is being reduced.

As RDS is only available in Internet Explorer, browser compatibility is no issue. It never made it to a standardized core technology in browser-server communication so the other browser makers did not try to adapt the approach. Moreover, Microsoft has announced it will discontinue the RDS support. So beyond Internet Explorer 7 RDS objects in the corresponding MDAC framework may no longer be supported.

In contrast to the disappearing RDS technology, the new W3C DOM Level 3 Load and Save Specification is just getting underway. Until now, only Opera supported some of its specification, but typically a new W3C web standard takes a bit of time to establish itself in the different browsers. Therefore, the forecast for this approach with respect to browser compatibility is fairly good, but until now there is no complete support at all.

5.5 Performance

The performance measurements show two characteristics. The first is that within one browser every approach takes a similar amount of execution time. The other characteristic is that between browsers, independently of the approach, the execution times are quite different. So it seems that the browser implementation itself is the responsible factor for performance independent of any of the approaches.

These characteristics were found in the loading performance, and also can be identified for saving as well, although less significant. For RDS and DOM3 LS only some measures could be captured due to the limited functionalities of those approaches. Thus, a meaningful comparison of these approaches to AJAX and HTML Inline Frames with respect to performance is not really possible.

5.6 Security Constraints

There are two types of security constraints that have to be differentiated. One type emerges from a typical aspect of asynchronous communication concerning recognition and control of actions performed by the application. This issue is not fixed to a specific approach. The second type is approach-specific, as dependencies on technical aspects of implementation. After the exclusion of self-developed security vulnerabilities as the allowing of SQL injections for example, both Inline Frames and DOM3 LS, are secure approaches. With RDS the necessary enabling of ActiveX opens the door to the client system for attacks. The AJAX implementation in current versions of Internet Explorer have to deal with the same issue, but this will be gone in the next version which does not need to enable ActiveX to instantiate an XHR object.

6 Influence to Web Applications

Since the introduction of “Web 2.0” in 2004 by the companies O'Reilly Media and MediaLive International³ as name for a conference the term made constantly headlines. In Tim O'Reilly's, founder of O'Reilly Media, definition of Web 2.0 beside very interesting approaches, such as the abolishing the separation between producers and consumers of web resources or mixing of user provided data or rather services, to the so called Mashup applications as core competences, one part is especially important with respect to this paper:

“Web 2.0 applications are... going beyond the page metaphor of Web 1.0 to deliver rich user experiences.” [13]

The page metaphor of Web 1.0 is the concept that every resource is accessible through a specific URI. Changing the displayed resource is done by changing the URI. Web 1.0 applications typically link several resources to allow a more or less convenient navigation through the provided information. The resulting page-based chain is broken up in Web 2.0 applications by using asynchronous communication and DHTML. After an initial load of a resource, arbitrary other resources can be loaded asynchronously as well as embedded to the web page without changing the actual URI. Logical chains can be performed without interruptions in one web page to achieve rich user experience. Thus, asynchronous communication plays a decisive role in achieving this rich user experience. Technologies for asynchronous communication, as AJAX, can be used to build highly

³ In the meantime MediaLive International became CMP technology

interactive user interfaces and low-latency behavior in web applications, which are full-fledged compared to traditional desktop-applications. Such applications often are categorized as Rich Internet Applications and belong to Web 2.0.

But new technologies and applications usually also create problems. Based on years of usage people have very precise, even though subconscious, expectations about how web applications should behave. Drag and drop functionalities, forms without submit button, or unsolicited dynamically changing content for example are not expected. Users are familiar with standardized user interfaces, latencies and web page reloads, the instead provided rich user experience of Web 2.0 applications might seem pretty odd for them. The problem of recognizing the beginning and ending of processes within an application is extended by making errors in such processes not noticeable for the user. If the user guidance of an application encourages the user to attend to one process while a former initialized process is still running in background, it has to be assumed that the user's attention is narrowed to the display area of the current process. State changes or errors arising from an unobserved, possibly assumed as already finished, process will irritate and annoy users. Developers are advised to implement status indicators so that the user can identify and follow processes and progresses.

7 Conclusion

A brief characterization of the non-AJAX approaches, concludes that Inline Frames were not originally meant for asynchronous communication, RDS is too specialized as well as being nearly dead, and DOM3 LS is still in the fledgling stages, reveals that a comparison has to focus on common applicability. With respect to this common applicability, the positions emerged by discussing implementation issues, features, and drawbacks became solid in doing the criteria-based comparison for the approaches. AJAX outperformed its competitors in the comparison by a consistent good performance in all decisive criteria, which likewise led to the assessment that AJAX is the recommended approach to implement almost all elaborated use-cases in web applications.

The increasing demand for asynchronous communication and the intrinsic capabilities of AJAX for implementing such communication without constricting the usability by having the shortcomings of the other approaches makes AJAX come to the fore in web development. Especially current Web 2.0 applications use the approach and thereby are advertisers for it. As, in the author's opinion, the evolution to Web 2.0 will proceed, AJAX will obtain more and more importance. Future web applications will not only appear as nowadays desktop applications, but will replace them in many areas.

References

- [1] Jesse James Garrett, *AJAX: A New Approach to Web Applications*, Article on the Adaptive Path Website, Published in 2005, <http://www.adaptivepath.com/>

publications/essays/archives/000385.php, Last visited 5th May 2006

- [2] Tom Noda, Shawn Helwig, *Rich Internet Applications*, Best Practices Report from UW E-Business Consortium, Published in 2005, <http://www.uwebc.org/opinionpapers/docs/RIA.pdf>, Last visited 5th May 2006
- [3] Google Inc, *Google Maps Website*, <http://maps.google.com/>, Last visited 25th June 2006
- [4] Sam Stephenson, *Prototype.js Framework Website*, <http://prototype.conio.net/>, Last visited 25th June 2006
- [5] Dojo Foundation, *Dojo Toolkit Website*, <http://dojotoolkit.org/>, Last visited 25th June 2006
- [6] Christian Gross, *Ajax Patterns and Best Practices*, Apress Inc, Berkeley, USA, 2006
- [7] W3C Recommendation 24th December 1999, *HTML 4.01 Specification*, <http://www.w3.org/TR/html4/>, Last visited 3rd July 2006
- [8] Apple Developer Connection, *Remote Scripting with IFRAME*, Published in 2002, <http://developer.apple.com/internet/webcontent/iframe.html>, Last visited 3rd July 2006
- [9] J. Papa, *Professional ADO 2.5 RDS Programming with ASP 3.0*, Wrox Press Ltd. Birmingham, UK, 2000
- [10] P. Shirolkar MSDN Library Article, *Data Access Technologies Road Map*, Published in 2002, Revised in 2005, http://msdn.microsoft.com/data/ref/mdac/default.aspx?pull=/library/en-us/dnmdac/html/data_mdacroadmap.asp, Last visited 26th July 2006
- [11] W3C Recommendation 7th April 2004, *DOM Level 3 Load and Save Specification*, <http://www.w3.org/TR/DOM-Level-3-LS/>, Last visited 3rd July 2006
- [12] Opera Software ASA, *Web Specifications Supported in Opera 9*, <http://www.opera.com/docs/specs/opera9/>, Last visited 3rd July 2006
- [13] Tim O'Reilly, O'Reilly Radar Website (Blog), *Web 2.0: Compact Definition?*, Published in 2005, http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html, Last visited 12th September 2006