Bachelor Thesis

RICH INTERNET APPLICATION DEVELOPMENT

Markus Moldaschl
0751916

Thesis advisor: ao. Univ.-Prof. Dr. Rony G. Flatscher
Institute for Management Information Systems
Vienna University of Economics and Business Administration

Abstract

This Bachelor thesis introduces the reader to basic concepts of Rich Internet Application (RIA) development. It outlines why traditional web applications fail to live up to expectations of today's web clients and describes how Rich Internet Applications excel in terms of user experience by closing the gap between the web and the desktop conglomerate.

Furthermore, the project presents selected, leading edge RIA development frameworks. On the basis of AJAX, Adobe Flex, Microsoft Silverlight and JavaFX distinctive key functionality is discussed in great detail and benefits and weak spots along with suggested areas of application are highlighted.

In conclusion, the thesis briefly depicts HTML 5 and meets concerns of the inescapable question, whether HTML 5 will have a significant impact on proprietary RIA solutions or even make them obsolete or not.

Keywords: RIA, AJAX, JavaScript, Adobe Flex, Flash, AIR, Microsoft Silverlight, JavaFX, Java, Java Scripting, BSF, BSF4ooRexx, HTML 5

Table of Contents

Figures

Listings

Tables

1 Introduction

This very first chapter deals the motivation behind this project work and the goals set. Additionally, it outlines the structure of the thesis and the topics discussed in the respective chapters.

1.1 Motivation and Goal

The World Wide Web, although based on the grand vision of Tim Berners-Lee, used to be a static collection of interlinked text documents to a large extent. The web rapidly proliferated but left potential untapped. In fact, traditional web sites could not serve the dynamic demand for collaborative work, adequate distribution of information, e-commerce and entertainment. The interaction model web sites implemented interfered with working behavior users adopted from the desktop conglomerate. Rich Internet Applications basically achieve to get rid of the document-centric “click, wait, refresh” user interaction paradigm.

Usability has become a central concern in software development. Companies often fail to engage customers with the online representation of their business. Customers choose the vendor’s site which allows them to accomplish their tasks as effective and reliably as possible.

Equipping your online presence with tools that act as intelligent assistances shielding the user from complexity and providing context sensitive information in a transparent way is the key to higher conversion rates, increased sales, longer stays on sites, more frequent repeat visits and deepened customer relationships. The Holy Grail is to provide seamless, focused, connected and aware experience contributing to a maximal degree of user satisfaction.

Web applications evolve and adopt (and even improve) artifacts of desktop applications. RIAs are ubiquitous; think of trailblazers like Amazon or Google Maps. Today various mature RIA development frameworks exist. They provide APIs for designing user interfaces more efficiently than with plain old GUI libraries like Java Swing; the outcome is more expressive. The goal is to design fully fledged software systems with sophisticated graphical user interfaces integrating with data stores and remote web services.

The main goals of this thesis are to:

- Introduce Rich Internet Applications, describe how they work, how beneficial they can be and how they close the gap between the Web and the desktop.
- Discuss major RIA development frameworks and impart knowledge about their functionality.

The theoretical part is supported by plenty of nutshells operationalizing presented concepts and exemplifying certain functionality.

1.2 Thesis Structure

The motivation and goals and the roadmap of the project work are presented in chapter one. Chapter two outlines the evolution of web applications from simple text representations to fully functional software systems. It describes the functional asset and the benefits of RIA, highlights characteristics web applications should obtain in order to satisfy the needs of clients, depicts how RIAs can contribute to business goals and categorizes RIA technologies.

After basic information is provided, the trailblazer in RIA development, AJAX, including its mechanism for asynchronous communication with the server – the heart of every RIA, is presented in chapter three. Analogous to AJAX, the leading edge technologies from the plug-in based technology conglomerate, Adobe Flex, Microsoft Silverlight and JavaFX, are covered over the chapters four to six. Subject to discussion are language characteristics and remote service handling while placing special emphasis on the interaction with the DHTML conglomerate (especially with JavaScript) and ultimately with other RIA scripts. Desktop (out-of-browser) capabilities are presented additionally, if available.

Chapter seven tries to highlight the strengths and weaknesses of every single studied development framework and tries to give general recommendations on use cases.

An outlook on HTML 5 and its expected features is presented in chapter eight. This chapter also deals with the manifest and latent impact of HTML 5 on the studied proprietary technologies. The last chapter concludes the thesis with a short compendium of the lessons learned in the previous chapters.

2 Overview of Rich Internet Applications

The term Rich Internet Application (RIA) is neither standardized nor is referring to a specific technology but rather states an umbrella term for applications which are intended to fill the gap between common web and desktop applications. An intuitive flexible graphical user interface allowing user gestures like drag and drop and the ability to retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page are typical characteristics of a RIA.

This chapter discusses standard web applications and their limitations and how to overcome them. It describes the core functionality of a RIA, e.g. the ability to take over responsibility for presentation and (re)calculation processes on the client computer. Furthermore, RIAs are sophisticated solutions for challenging e-commerce, support process automation and contribute to overall business goals.

2.1 The Evolution of Web Applications

The Internet was originally designed for simply transporting documents and information. Simple web sites were composed of text documents interconnected through hyper-links. The main purpose of this static collection of HTML web pages was to display preformatted text. As the traditional Web relied heavily on a few interface controls, like combo boxes, buttons and forms it offered only a low degree of interactivity.

Communication with a server in the course of requesting or updating data is limited to synchronous HTTP calls resulting in a full page refresh. The bulk of business logic is placed in the middle tier (the server). In typical implementations, the client elements of traditional browser-based applications are limited to the interface logic (for example, HTML) with small amounts of script code (such as JavaScript) for minor data validation and control logic [DVPG05].

Classic web applications are often preferred for the following reasons:

- Standardized tags/scripts are easy to develop.
- No installation or updates necessary.
- Applications are accessible from networked computers.
- Applications can run on different operating systems.
- User interface (UI) is simple and standardized.

Nevertheless, seamless interaction has never been the strong suit of HTML applications. Since a full-page refresh is usually needed to change any part of the display, the ability of HTML-based applications to offer responsive feedback needed to deliver a truly seamless experience is limited [Mull03].

Enhancing the Functionality by Introducing the Application Server

As web sites grow, it becomes very hard to separate presentation from business logic in a 2-tier model, and the applications thus become cumbersome and hard to modify with regard to scalability and maintainability. In order to increase quality attributes such as security, reliability, availability, and scalability, as well as functionality, most of the software has been moved to a separate computer, the application server.

Separating the presentation (typically on the web server tier) from the business logic (on the application server tier) makes it easier to maintain and expand the software both in terms of customers that can be serviced and services that can be offered. Customers expect to be able to use web sites without training. Thus, the software must flow according to the users' expectations, offer only needed information, and when needed, provide navigation controls that are clear and obvious [OfJe02].

Though there has been significant progress in server-side implementations, the means of rendering information to the end user remained the same. Further, the inability to handle complex multi-step business workflows was another problem. The synchronous nature of traditional Internet applications forced back-and-forth page-flips on users, resulting in a slow and confusing user experience [DBBO07].

The need to incorporate the rich, interactive and responsive features of desktop applications was felt increasingly.

Desktop applications

In contrast to (classic) web applications desktop applications offer the following advantages:

- Richer user experiences through immediate and accurate feedback.
- No page reloading necessary to see the results of user interaction.
- Support both online and offline working.
- Great variety of flexible and intuitive interface controls.

On a downside, traditional applications are tethered to your computer. Often, they are tethered to your operating system, and the file system the operating system exposes. They require installations that are long and cumbersome, and almost all the content they generate or consume is stored locally on the computer, making it difficult to share with others [Treto08].

The Rich Internet Application (RIA) addresses the best elements of both, standard Web and desktop applications as depicted in figure 1.

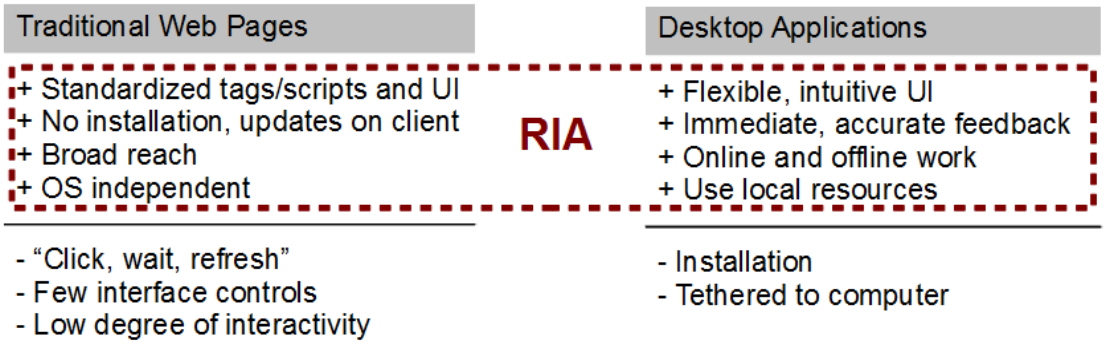


Figure 1: RIA Combining Advantages from the Web and the Desktop Sphere

RIAs are designed to enable the Web to evolve beyond the page based, document-centric metaphor commonly associated with the browser approach. Figure 2 shows the evolution from web application to RIA and the synergies it combines.

Figure 2: Evolution of Web applications [DBBO07]

A caricature in figure 3 illustrates this RIA evolution in the context of the evolution of humanity.

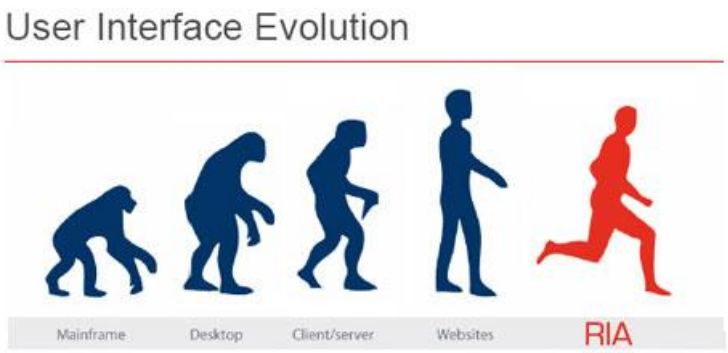


Figure 3: Caricature on the RIA Evolution[1]

[1] http://phptrainingcourses.com/ria-php-ajax-training-institute.html – requested in February 2011
[2] Adobe Systems acquired Macromedia on December 3, 2005 and as of 2010 controls the line of Macro-

2.2 Functionality and Characteristics of RIAs

Macromedia[2] first coined the term Rich Internet Application in 2002 and defines it as combining the best user interface functionality of desktop software applications with the broad reach and low-cost deployment of Web applications and the best of interactive, multimedia communication. The end result: An application providing a more intuitive, responsive and effective user experience [LRWA08].

Another definition puts emphasis on the shift from a thin client to thick client architecture, designed not only to render data from the server but also to process data corresponding to user interaction on the client [BuKo09]:

> "Rich Internet Applications (RIAs) are web applications, which use data that can be processed both by the server and the client. Furthermore, the data exchange takes place in an asynchronous way so that the client stays responsive while continuously recalculating or updating parts of the user interface. On the client, RIAs provide a similar look-and-feel as desktop applications and the word 'rich' means particularly the difference to the earlier generation of web applications."

2.2.1 RIA Architecture

Regardless of the technology framework used to build a RIA the underlying architecture is basically the same. A web application usually consists of a three layer model. The first layer hosts the data access, the second the business logic, and the last one the presentation logic in analogy to the MVC pattern[3]. As mentioned before, with a traditional web application the browser engine is only responsible for rendering the provided data from the server according to the HTML markup as all three layers – presentation layer, business layer and data layer (plus an optional service layer) reside on the server. In other words, the client's web browser is qualified for just displaying the data it receives from the server.

---

[2] Adobe Systems acquired Macromedia on December 3, 2005 and as of 2010 controls the line of Macromedia products, e.g. Flash.

[3] http://en.wikipedia.org/wiki/Model-view-controller – requested in November 2010

In a RIA architecture the client's browser takes over responsibility from the server as the whole presentation layer and part of the business logic layer move to the client. Thus, the client's browser is aware of the data it needs from the server. As a consequence the client's browser only requests essential data without extra markup which leads to less complicated server requests and an overall decrease in server load.

A prerequisite for this intelligent processing is the existence of some kind of MVC on each side. The client MVC manages interaction between the user and the interface, requests data from the server and controls how data is presented in the view. The MVC on the server handles requests from the client [Tric08].

Figure 4 provides a holistic view on the layered model. RIAs support rich graphics and streaming media scenarios, while providing most of the deployment and maintainability benefits of a web application.

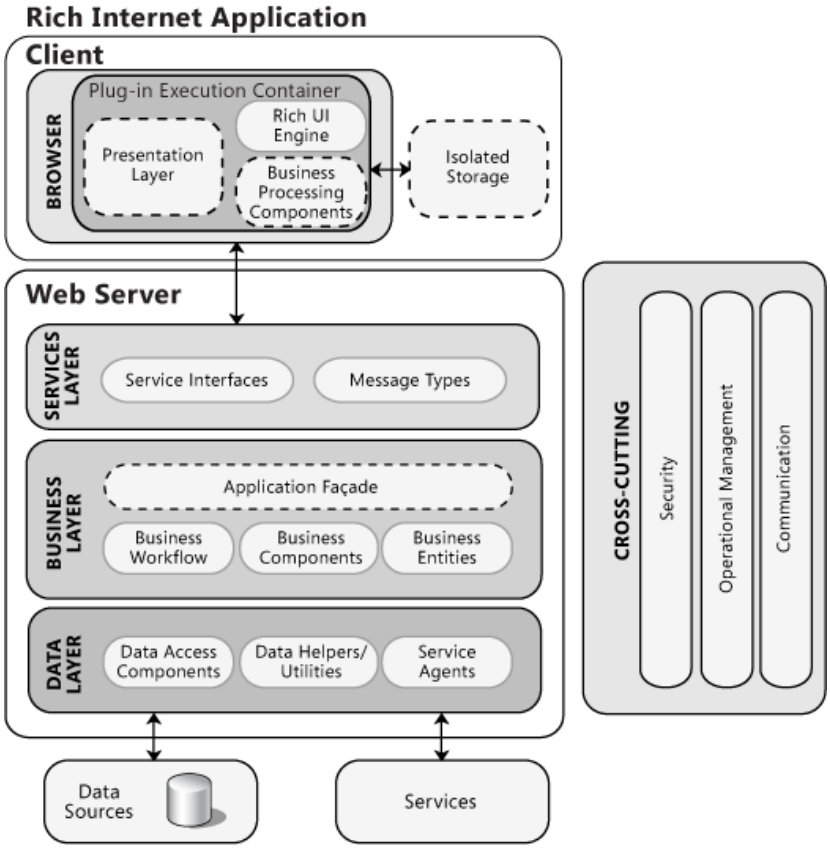


Figure 4: Typical RIA Architecture[4]

[4] http://msdn.microsoft.com/en-us/library/ee658083.aspx – requested in November 2010

RIAs may run inside a browser plug-in and utilizes a web infrastructure combined with a client-side application that handles the presentation. The plug-in provides library routines for rich graphics support as well as a container that limits access to local resources for security purposes. The services layer processes the requests from the client application and delegates actions on the server. This could be saving data in a database, updating the file system, some kind of analytical processing, or returning chunks of data to the server. The big differentiation here is that there is no user interface. Instead of a user interface, the view would be the format of the data that is being returned to the client application.

2.2.2 Main Characteristics

One of RIA's distinctive features is that it consists of a GUI which is able to aggregate and visualize complex data from different sources for intuitive manipulation by the client. Other characteristics are the asynchronous nature of communication with the server and the ability to occasionally leverage the volatile and/or persistent resources on the client. This section discusses the elemental features of a RIA.

Enhanced Graphical User Interface (GUI) Behavior

By leveraging elements from desktop GUI widgets the user interaction becomes more flexible and intuitive. Many RIA technologies enable the capability to manipulate graphics onscreen at runtime. This enables charting and complex data visualizations, and in some cases, even 3D modeling. In addition, intricately related data from different source can be aggregated on a single screen, which eases complexity and supports decision-making. A RIA can also integrate multimedia and animations, e.g. transitions to make task steps visible for the user, and facilitate the use of drag & drop and context menus.

Front end use

Service-Oriented Architecture (SOA)[5] has gained acceptance as an architectural paradigm to organize and manage enterprise resources as loosely coupled services that can be orchestrated to business processes. While enterprise business logic

[5] http://en.wikipedia.org/wiki/Service-oriented_architecture – requested in November 2010

and data are exposed as business and data services using SOA, RIA helps these services to reach their intended end users in an interactive and intuitive way.

Responsiveness and interactivity

The client interface maintains state of the application completely on the client side. It does not rely on the server to maintain information about what the current user is doing. This enables simplifying back end logic to be more service oriented and asynchronous. The user interface also does not need to wait on the back end logic.

Asynchronous data services are performed behind the scenes, while the user interface is still accessible to the user. The user may not even know that a remote service method has been invoked. This enables the user to keep using the application seamlessly, which helps lend to the desktop-application feeling.

Additionally, asynchronous data services typically only contain data, no UI declarations; it is up to the stateful client to determine how the data will be displayed. This typically reduces the size of the requests back and forth to the server, and often results in reduced server load and better interface responsiveness. [Treto08].

Data Distribution

The developer can decide about the distribution and even design an application that may temporarily be used irrespective of the server. Therefore, a RIA can use the client's persistent and volatile resources. This allows offline usage of the application Data can be cached and manipulated on the client, and finally sent validated, formatted and prepared to the server once the operation has been completed.

2.3 Emphasize the User Experience (UX)

The usability of a design determines how well the users will be able to perform the supported tasks. The best designs support the task in a way that is simple and natural for the intended user. For an interactive experience this results in a design that lets users engage the application directly.

The interface itself becomes transparent and lets the user work without thinking about the complexities of how. It allows the user to focus exclusively on the data being manipulating and his/her progress toward the goal [Mull03].

A RIA may feature the following characteristics to provide an optimal degree of user satisfaction:

- Seamless

  Interactive software produces a seamless user experience when it provides immediate responses and smooth transitions between tools, modes, states, displays, and other focal points within the application. Context is preserved wherever possible and transparently restored when necessary. The users' ability to maintain focus on the work they are trying to accomplish maximizes their performance and satisfaction with the tool.

- Focused

  A focused experience has a purpose that is clearly defined at the outset and continuously reinforced. Monolithic applications are superseded by SOA modules which support a narrowly defined set of tasks and information. Moreover, tools that target a single task are easier to learn, to remember, and use efficiently.

- Connected

  Connected applications will transparently locate and exchange information with a variety of data sources and communication devices to shield users from the complexity of having to manage the connections themselves. Applications will connect to other applications and remote data sources to transparently exchange information and facilitate users' tasks.

- Aware

  Applications seem to be aware of what the user is trying to do when they recognize the current operating context (location, goals, tasks, applications) and use this information to transparently facilitate user and task needs.

  Applications that dynamically adjust their behavior to reduce the user's workload quickly come to be seen as intelligent assistants rather than just tools. Filtering their interface elements based on a user and task context can help designing a workflow which requires repetitive user conformation more efficiently.

2.3.1 RIA as Part of Process Automation

In fact, RIAs help Business Process Management (BPM) overcome one of its greatest challenges: the gap between an organization's ability to automate processes and users' readiness to participate in them. As business managers know all too well, failure to engage staff and customers fully in processes can result in high attrition, low conversion rates, overloaded customer service centers, lengthy cycle times and missed opportunities - all things that RIAs can help address [Wick08].

Streamlined business processes encapsulated in a role-sensitive RIA transform processes as well as customer experiences and employee productivity by pulling together all required information into a view that matches the steps of each task.

Creating a user-centric application delivers a variety of substantial business benefits including: higher conversion rates, increased sales, increased brand loyalty, longer stays on sites, more frequent repeat visits, reduced bandwidth costs, reduced support calls, and deepened customer relationships.

2.4 Types of RIAs

Basically Rich Internet Applications can be differentiated by how they integrate with the browser. They leverage the scripting engine, use a separate runtime or operate standalone without internet connection. The following three categories exist [HaGo07]:

Browser-based solutions: Developers deliver rich Internet applications using a browser's document object model and scripting engine, along with Ajax frameworks. Browser-based RIAs install and execute automatically, as long as the client's browser has JavaScript enabled.

Plug-in-based solutions: Using a browser plug-in as a common runtime, developers can provide the same ability to run across operating systems as a browser-based solution, but without the "headaches" that individual browsers create for RIAs. This approach minimizes cross-browser testing issues, but users may not be willing to update plug-ins or wait for a lengthy install process to complete if they don't already have the latest version.

Desktop-based solutions: The third option is to install a set of runtime components on the desktop client outside of the browser's security constraints. Desktop-based solutions provide access to local file systems and the ability to persist information, even when web connection is interrupted.

2.4.1 Technology study

The aforementioned RIA categories – browser-, plug-in- and desktop-based applications – are subject to discussion in greater detail over the next four chapters. They will be analyzed on the basis of the leading development frameworks AJAX, Adobe Flex, Microsoft Silverlight and JavaFX. These frameworks are chosen for demonstration because of the broad support of their runtimes (Flash Player and .NET CLR) and/or due to the fact that they leverage widely accepted (standardized) technologies like JavaScript and Java.

The study focuses on the following functionality:

- Language characteristics
- Remote services handling and integrating response data into application
- Interaction with environment and other scripts
- Desktop/out-of-browser capabilities (if provided)

This paper is not meant to discuss GUI building capabilities in great detail as all four technologies (AJAX with the support of widget toolkits) feature a tremendous rich set of building block libraries. Nevertheless, language specific aspects, e.g. the efficient way of creating a GUI with a declarative language and outstanding GUI components are demonstrated and discussed.

RIA Proliferation

Figure 5 and figure 6 concern the proliferation of the studied technologies. Based on the number of job listings aggregated on `http://www.simplyhired.com/` as shown in figure 5, AJAX is on the cutting edge when it comes to RIA development.

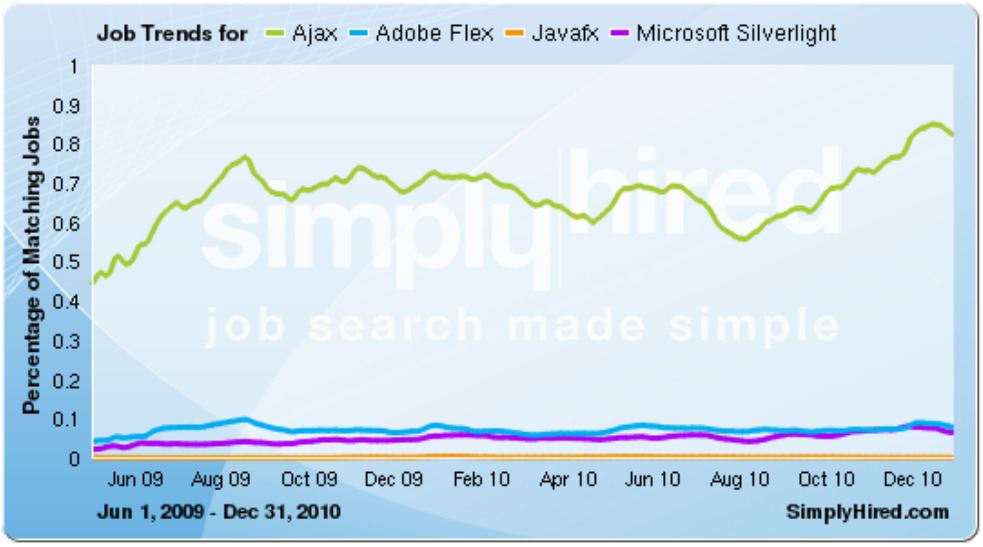


Figure 5: Job Trends Concerning AJAX, Flex, JavaFX and Silverlight[6]

Let's have a closer look on the three plug-in based technologies. The chart in figure 6 offers clues how Flex, JavaFX and Silverlight compete based on relative market penetration in the period from January 2010 to December 2010.

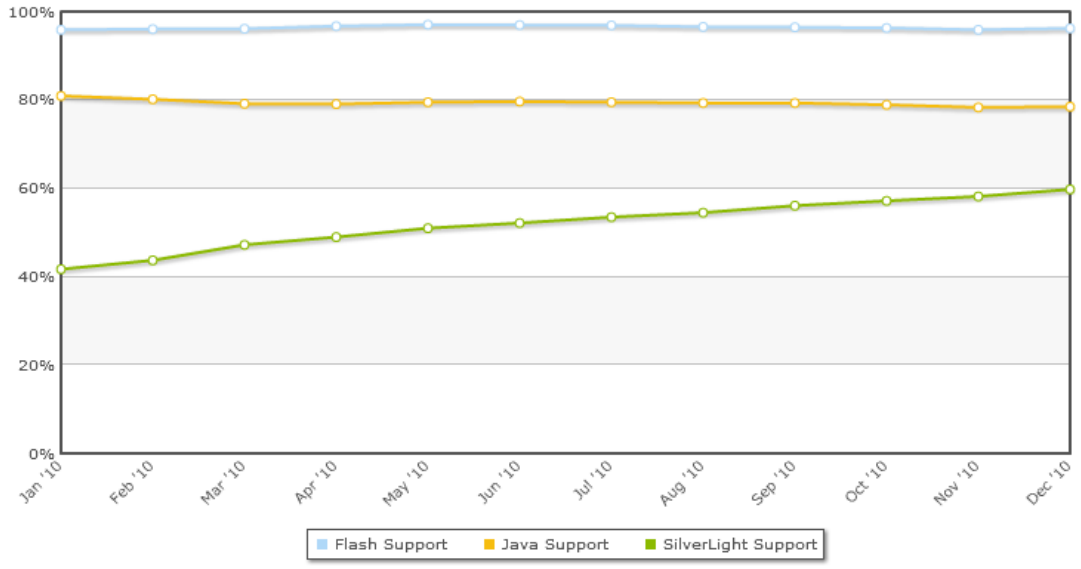


Figure 6: Market Penetration and Usage of Plug-in Based Technologies[7]

[6] http://www.simplyhired.com/a/jobtrends/trend/q-ajax%2C+adobe+flex%2C+javafx%2C+microsoft+silverlight – requested in December 2010
[7] http://www.statowl.com/custom_ria_market_penetration.php – requested in December 2010

Figure 7 is an extract from `http://riastats.com/`. This website detects which plug-ins are installed on the visitor's system and displays the results. The site itself uses the Flash Player plug-in, just as an aside. According to the gathered numbers, roughly 95% of the client machines have the Flash Player installed, 75% support Java technology while 70% feature a Silverlight plug-in as of December 2010.

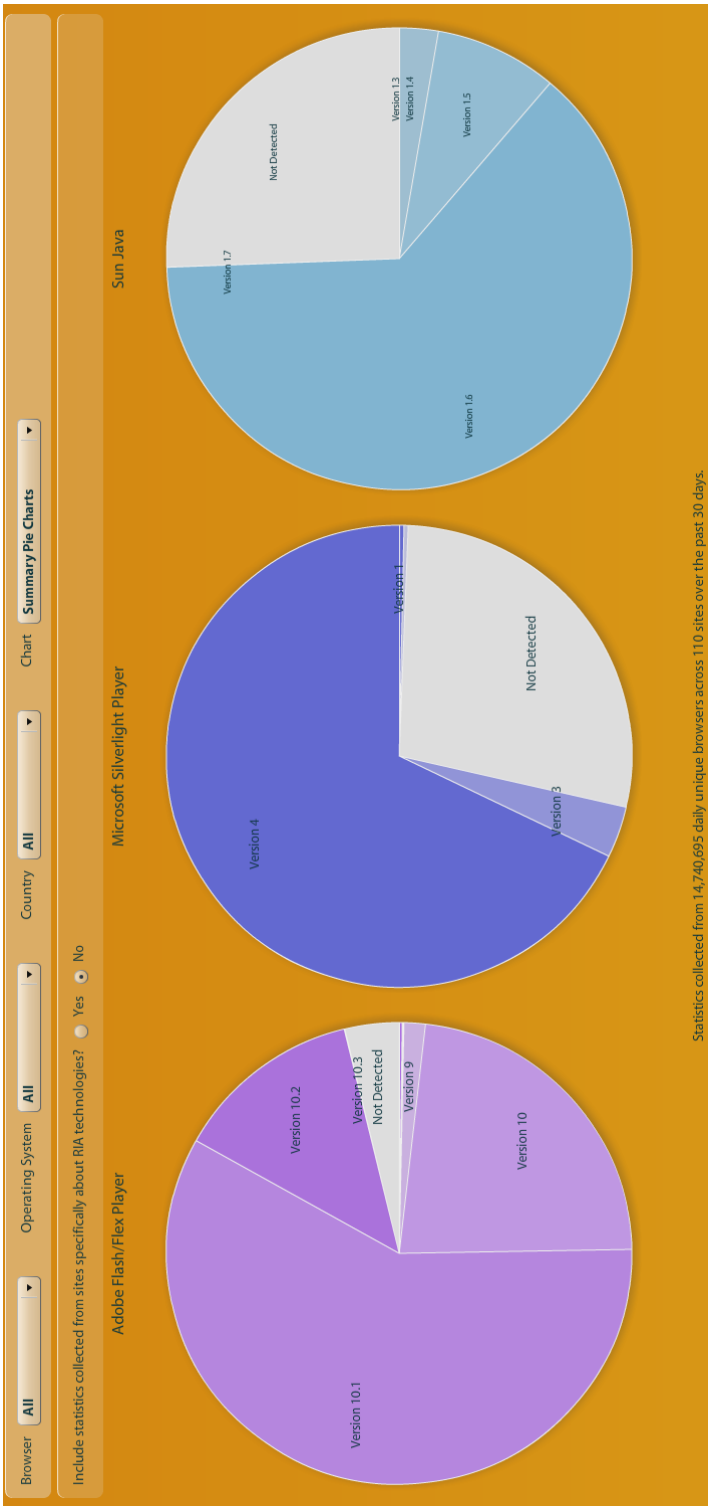


Figure 7: Plug-in Support Detected on Systems of Visitors of http://riastats.com/

Deployment

In terms of a categorization illustrated in table 1, AJAX is the only browser-based technology purely leveraging standards that are part of web application development for many years. Applications built with one of the three other frameworks use a separate plug-in to run in the browser.

	Browser-based	Plug-in-based	Desktop-based
AJAX	✓		
Adobe Flex		✓	✓
Microsoft Silverlight		✓	~
JavaFX		✓	✓

Table 1: RIA Technologies Deployment

With Adobe Flex and the Adobe Integrated Runtime Environment (AIR) it is also possible to build applications which achieve to overcome certain restrictions web applications are facing and look and feel more like desktop applications. For example, an AIR application is able to leverage the file system of the client machine and is able to operate independently of the server offline[8]. Microsoft's out-of-browser (OOB) approach does not build on a separate runtime environment; it still uses Silverlight. Every Silverlight application can simply be configured for OOB. That is why, the OOB approach appears to be similar to AIR but it has more restrictions imposed than AIR applications have[9]. In fact, Silverlight OOB applications are isolated from the local file system – they are not allowed to interact with any data not part of the application on the client machine. So the Silverlight framework is marked with a tilde in the category "Desktop-based" as a consequence. JavaFX applications can undock from the browser and camouflage as desktop applications[10]. A JavaFX application deployed as Java applet and installed on the client machine can leverage the proven Java security model to access local resources as long as Java SE Version 6 Update 10 – released on October 15 2008 – is installed.

[8] See chapter 4.5 for detailed information on Adobe Integrated Runtime (AIR).
[9] See chapter 5.5 for detailed information on Silverlight OOB and a brief comparison with AIR.
[10] See the section "Undock from Browser" in chapter 6.1.3 for information on how to drag a JavaFX application onto the desktop.

3 AJAX

AJAX builds on the standard repertoire of common web application development, namely HTML and JavaScript. Early efforts in asynchronous communication with the server were achieved by using hidden forms with width and height of zero pixels and later on with the `<iframe>` tag. The frame served as a communication channel as response according to submitted form data could be extracted from it. As a result only the hidden frame and not the whole page had to be updated.

Great advantage was that such an “iframe” could be created on-the-fly using JavaScript and the DOM. This made asynchronous calls possible without having to plan a hidden frame in the HTML design (which encourages separation of code and presentation).

3.1 Basics of AJAX

The name itself is an acronym for Asynchronous JAvaScript and XML, which already contains two technologies and a technique for loading information. The name was coined by Jesse James Garrett. AJAX is not a single new technology, but a combination of standard technologies including HTML, CSS, JavaScript, XML and DOM which together with the `XMLHttpRequest` object achieve web application richness.

AJAX applications work unconditionally in browsers without the need to install any plug-ins. However, JavaScript implementations are not standardized across browsers and operating systems. This means that programs must be aware of the browser, the operating system, and their respective versions. Programs must execute different code depending on specific combinations thereof.

3.1.1 The AJAX Engine

At the heart of the AJAX method of communication with the server lies the AJAX engine. This is nothing more than some JavaScript code that instantiates and uses the `XMLHttpRequest` object.

Instead of loading a webpage, at the start of the session, the browser loads an AJAX engine — written in JavaScript and usually tucked away in a hidden frame. This engine is responsible for both rendering the interface the user sees and communicating with the server on the user's behalf. The AJAX engine allows the user's interaction with the application to happen asynchronously — independent of communication with the server. So the user is never staring at a blank browser window and an hourglass icon, waiting for the server to respond (cf. figure 8 and figure 9).

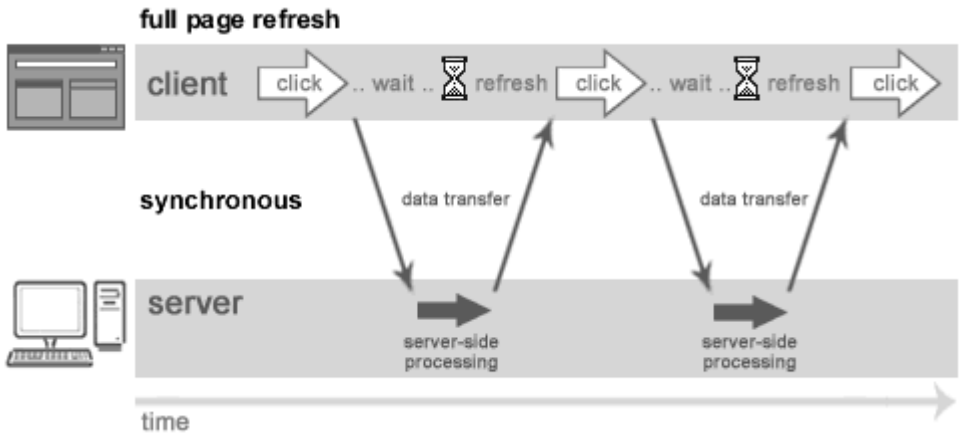


Figure 8: Classic Synchronous Wait-Refresh-Cycle [DeveAJ]

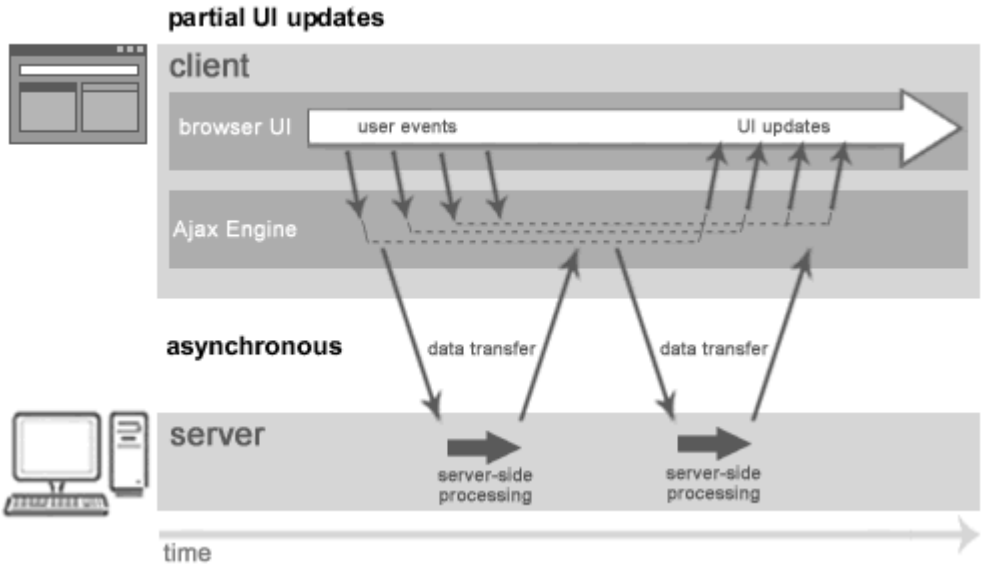


Figure 9: Partial UI Updates and Asynchronous Communication with AJAX [DeveAJ]

Every user action that normally would generate an HTTP request takes the form of a JavaScript call to the AJAX engine instead. Any response to a user action that does not require a trip back to the server — such as simple data validation or some navigation — the engine handles on its own.

If the engine needs something from the server in order to respond — if it's submitting data for processing, loading additional interface code, or retrieving new data — the engine makes those requests asynchronously, usually using XML, without stalling a user's interaction with the application [Garr05].

3.1.2 DHTML

By extending the static nature of HTML with the client-side scripting language JavaScript and the Document Object Model a certain degree of interactive behavior of web pages can be achieved without the need to request a server-side page refresh. For example, page elements can be inserted or removed dynamically or applied visual effects from Cascading Style Sheets (CSS) can be altered with a JavaScript function.

JavaScript

Released in 1995 by Netscape and Sun, JavaScript is an implementation of the ECMAScript language standard. It is a prototype-based, object oriented, freely available cross-platform scripting language. Since all sophisticated web browser support JavaScript, it does not make AJAX applications require a plug-in and is the de facto standard for dynamic web page development. For example, the technology can cause a linked page to appear automatically in a popup window or let a mouse rollover change text or images.

DOM

The DOM, a W3C standard since 1998, is a platform and language-neutral interface that lets developers create and modify HTML and XML documents as sets of program objects, which makes it easier to design Web pages that users can manipulate. The DOM grants access to the elements of the currently loaded HTML page and to the attributes of the page's elements, by providing methods and properties to retrieve, modify, update and delete parts of the document including content, structure and style[11] [LRWA08].

[11] For further reading about the Document Object Model please see:
http://www.w3.org/DOM/ – requested in December 2010
http://en.wikipedia.org/wiki/Document_object_model – requested in December 2010

For instance, in listing 1 the DOM API is used to append a newly created form to an existing table and to alternate the used styles.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>DOM example</title>
        <style type="text/css">
            body {
                font-family: Verdana, Arial, sans-serif;
            }
            table {
                width: 550px;
                padding: 10px;
                background-color: #c5e7e0;
                border-style: solid;
            }
            td {
                padding: 10px;
            }

        </style>
        <script type="text/javascript">
            /* Manipulates the DOM to insert a form into the existing table.
               Removes some style properties and adds a new rule block*/
            function register()
            {
                var regform = document.createElement("form");
                var attr = document.createAttribute("id");
                attr.value = "regform";
                regform.setAttributeNode(attr);
                regform.appendChild(document.createTextNode("First name: "));
                var firstname = document.createElement("input");
                firstname.type = "text";
                firstname.name = "firstname";
                regform.appendChild(firstname);
                regform.appendChild(document.createElement("br"));
                regform.appendChild(document.createTextNode("Last name: "));
                var lastname = document.createElement("input");
                lastname.type = "text";
                lastname.name = "address";
                regform.appendChild(lastname);
                regform.appendChild(document.createElement("br"));
                var regbutton = document.createElement("input");
                regbutton.type = "submit";
                regbutton.value = "Submit";
                regform.appendChild(regbutton);

                var regtable = document.getElementById("regtable");
                var newrow = regtable.insertRow(1);
                var newcell = newrow.insertCell(0);
                newcell.appendChild(regform);

                var styles = document.styleSheets[0].cssRules[1].style;
                styles.removeProperty("background-color");
                styles.removeProperty("border-style");
```

```
                document.styleSheets[0].insertRule("#regform { \n\
                background-color: #c5e7e0; border-style: solid}", 3);
            }
            /*executed when all window components are loaded
            adds listener for click events on anchor element*/
            function doLoad()
            {
                document.getElementById('reglink').addEventListener('click',
                register, false);
            }
            // listens for the completed page load
            // equals to "window.onload()"
            window.addEventListener('load', doLoad, false);
        </script>
    <body>
        <table id="regtable">
            <tr>
                <td>some Text</td>
                <td>
                    <a href="#register" id="reglink">Register</a>
                </td>
            </tr>
        </table>
    </body>
</html>
```

Listing 1: DOM Manipulation of Elements Example

Figure 10 and figure 11 illustrate the initial state of the application and the state after the register link is clicked.

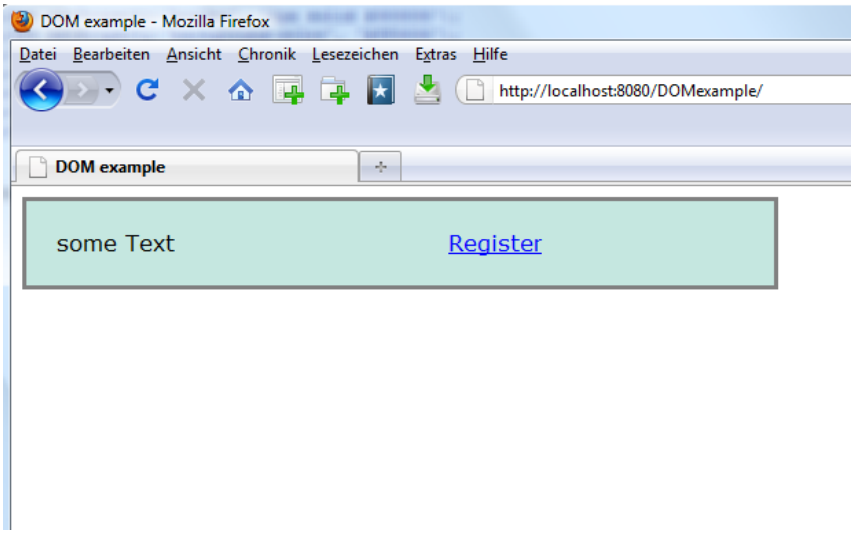


Figure 10: DOM Example: Initial State

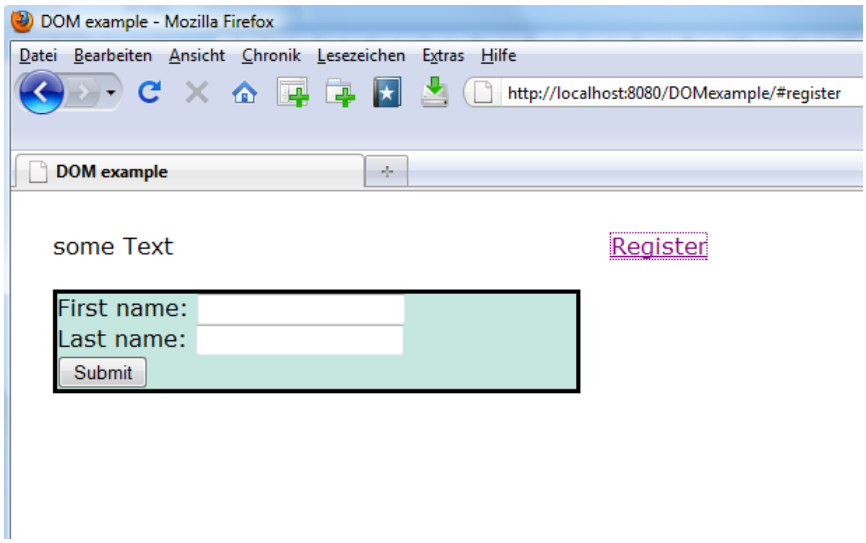


Figure 11: DOM example: Elements and Styles Manipulated

3.1.3 XML

XML is used to encode data for transfer between a server and a browser or client application. It is a markup meta-language that can define a set of languages for use with structured data in online documents. Any organization can develop an XML-based language with its own set of markup tags.

XML is recommended but not required for data interchange. That is why JavaScript Object Notation (JSON) is often used as an alternative format. Though, other formats such as preformatted HTML or plain text can also be used.

XML vs. JSON

One of the arguments for JSON is that it is lightweight in nature. On average, JSON requires less characters, and so less bytes, than the same data in XML. Because it uses JavaScript syntax, it requires less parsing than XML when used in AJAX applications because it is essentially serialized JavaScript objects. An object begins with "{" (left brace) and ends with "}" (right brace). Each name is followed by ":" (colon) and the name/value pairs are separated by "," (comma)[12]. Basically `eval()` can be used to parse JSON objects. However, `eval()` compiles and executes JavaScript code, that is why, it should only be used for trusted sources to

[12] http://www.json.org/ – requested in December 2010

avoid malicious code in input. A better alternative is `JSON.parse()`, which rejects all scripts and accepts only JSON text[13].

Nevertheless, XML works as a good data exchange format for moving data between similar applications. XML is designed to have a structure that describes its data, enabling it to provide richer information. XML data is self-describing. XML supports internationalization of its data. XML is widely adopted by the technology industry [Hold08].

3.1.4 XMLHttpRequest

The `XMLHttpRequest` object is the key for instant information updates from the server without any (visual) interruptions of the user experience. This is a JavaScript object that allows sending, receiving and processing HTTP requests to and from the server without refreshing the entire page. While first being implemented as an interface in Microsoft's ActiveX framework for Internet Explorer, it has later become a native JavaScript object [InterAJ].

Figure 12 illustrates a typical process flow of an AJAX application. The user triggers an event, which results in a JavaScript call to a function that initializes an `XMLHttpRequest` object via `var request = new XMLHttpRequest();` and respectively with Internet Explorer 6.x and older via:

```
var request = new ActiveXObject('Microsoft.XMLHTTP');
```

For a request to the server the methods `open()`, `send()` and the property `onreadystatechange` are used. For instance like this:

```
request.open('GET', URL, true);
request.onreadystatechange = callback;
request.send('');
```

With `open()` the request is prepared by assigning the method (either GET or POST), the URL and a flag for asynchronous communication. The property `onreadystatechange` listens for a state change of the request and calls the assigned function, which processes the server response. In case of a POST request optional data can be given as an argument to `send()`, which executes the request.

[13] cf. http://www.json.org/js.html – requested in March 2011

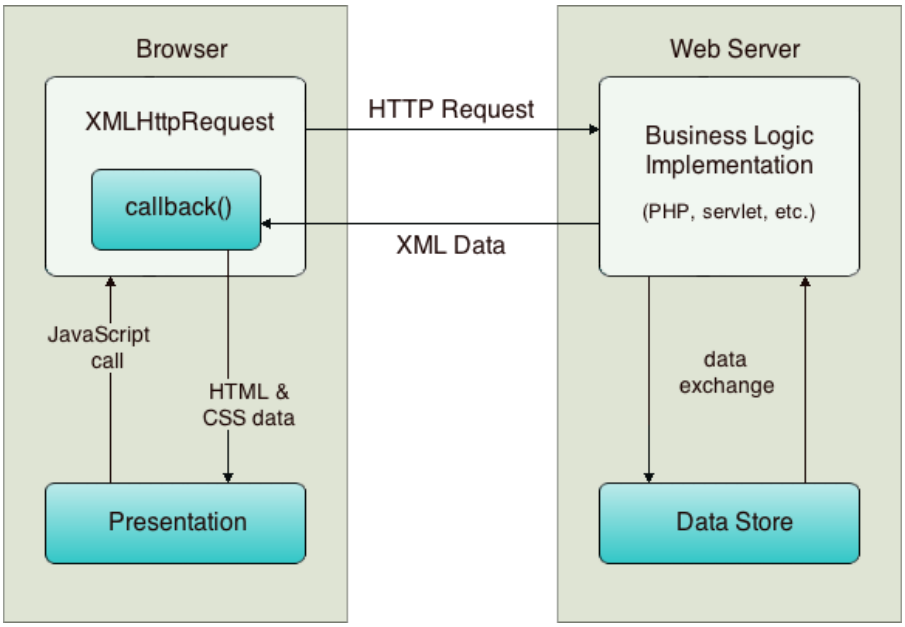


Figure 12: AJAX Process Flow[14]

On the web server, an object such as a servlet or listener handles the request. Data is retrieved from the data store, and a response is prepared containing the data in the form of an XML document. Finally, the `XMLHttpRequest` object receives the XML data using the callback function assigned to `onreadystatechange`, processes it, and updates the HTML DOM and consequently the view to display the page containing the new data.

3.1.4.1 Processing the Response

Once the response data from an `XMLHttpRequest` is received, it can be parsed and integrated into a HTML page with several DOM methods.

```
function parseResponse( ) {
    /* Is the /readyState/ 4? */
    if (request.readyState == 4) {
        /* Is the /status/ 200? Is the request successfully processed and
           will the output of the request included in the response?*/
        if (request.status == 200) {
            var response = request.responseXML;
            var paramList = response.getElementsByTagName('param');
            var out = '<ul>';
            for (i = 0, il = paramList.length; i < il;)
```

[14] http://netbeans.org/kb/docs/web/ajax-quickstart.html – requested in December 2010

```
                out += '<li>' + paramList[i++].firstChild.nodeValue + '</li>';
            out += '</ul>';
            document.getElementById('list').innerHTML = out;
        } else alert('Problem retrieving the data: \n' + request.statusText);
        request = null;
    }
}
```

Listing 2: Function for Processing Server Response Data

First the state of the request is checked using the rules depicted in table 2.

`readyState`	This property represents the current state that the object is in. It is an integer that takes one of the following: • 0 = uninitialized (The `open()` method of the object has not been called yet.) • 1 = loading (The `send()` method of the object has not been called yet.) • 2 = loaded (The `send()` method has been called, and header and status information is available.) • 3 = interactive (The `responseText` property of the object holds some partial data.) • 4 = complete (The communication between the client and server is finished.)

Table 2: XMLHttpRequest status codes (cf. [Hold08])

After the transmission is completed and the response was received, understood, accepted and processed successfully (cf. table 2) the XML data is parsed. The DOM of the received XML data is traversed node by node to construct an unordered list in the example from listing 2. It is crucial, that the client-side script is aware of the format and the structure of the server supplied data. In a last step, the list data is added to the `innerHTML` property of an element on the HTML page. The `innerHTML` property is a proprietary extension from Microsoft and automatically translates a given string into the DOM of an object. However, it does not check if the markup is valid or well-formed. `innerHTML` is W3C DOM compatible and works basically in combination with all modern browsers[15].

[15] See http://www.quirksmode.org/dom/w3c_html.html – requested in April 2011 – for `innerHTML` compatibility checks.

3.2 Digression: Web Services and Frameworks

This section aims to provide a theoretical background of web services and application frameworks essential for developing sophisticated RIAs. The reader is introduced to different types of web services architecture and web services standards used to transmit data from the server to the client and vice versa and to define interfaces for communication with the server and for invocation of provided services. Additionally, this section briefly discusses frameworks concerning how they ease application development.

3.2.1 Web Services

A Web service is in most cases an API on a remote system accessed via the Internet. Usually it is used to communicate information between systems by delivering a feed of data with a specific format, e.g. XML, to the requesting system.

Using a web service in an application has the potential to greatly reduce development time and speed up application deployment simply by being a resource to use, instead of having to create it all from scratch. [Hold08]

A script is restricted to communicate with resources from its own domain of origin. So, the developer is bound to his own server with his own services. But especially with mash-ups the intent is to aggregate data from a multitude of sources. Although Internet Explorer 8 introduced the `XDomainRequest`[16] property and more or less cumbersome workarounds exist, the most sophisticated solution is to use a proxy, for instance a server side PHP script or a Java Servlet that loads the requested service (cf. figure 13).

[16] http://msdn.microsoft.com/en-us/library/cc288060%28v=VS.85%29.aspx – requested in January 2011

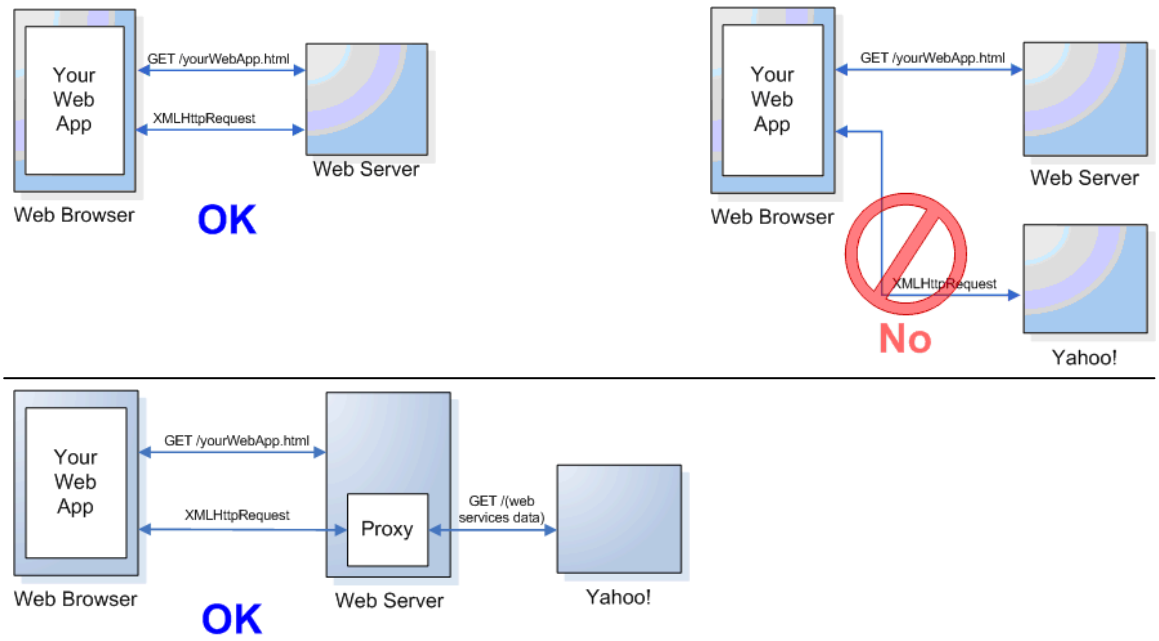


Figure 13: Using a Proxy to Obtain Cross Domain Access[17]

Furthermore, using such an intermediary frees the client from worrying about the web service's language and data format and, since the server script does the parsing, execution speed on the client improves.

3.2.1.1 Web Service Architectures

Web services are architected in different ways depending on whether the interface should be suited for a specific procedure or should be wrapped technological neutral or should use the HTTP paradigm. The common web services architectures are Remote Procedure Call (RPC), Service Oriented Architecture (SOA) and Representational State Transfer (REST).

Remote Procedure Call

Remote Procedure Call (RPC) architecture enables an application to start the process of an external procedure while being remote to the system that holds it. In simpler terms, a developer writes code that will call a procedure or subroutine that could be executed either within the same application or in a remote environment. The developer does not care about the details of this remote action, only the interface it begins to execute and the results of that execution [Hold08].

[17] http://developer.yahoo.com/javascript/howto-proxy.html – requested in January 2011

Service Oriented Architecture

In contrast to RPC which aims for technology neutral interface wrapping of services, Service Oriented Architecture (SOA) defines loosely coupled services emphasizing a high degree of reusability of business functions (code assets). These services communicate using a formal definition that is independent of the application's programming language and the underlying operating system. The individual services are accessed without any knowledge of their underlying resource dependencies.

Representational State Transfer

Representational State Transfer (REST) identifies resources via global identifiers (URI) and uses a restricted set of methods similar to HTTP. The requester of a service uses `GET`, `POST`, `PUT` or `DELETE` to retrieve representations of resource for manipulation.

Dr. Roy Fielding coined REST in his doctoral dissertation[18] as a collection of principles that are technology independent, except for the requirement that it be based on HTTP.

A RESTful system is identified by the following principles [CBB09]:

- All components of the system communicate through interfaces with clearly defined methods and dynamic, mobile, code.
- Each component is uniquely identified through a hypermedia link.
- A client/server architecture is followed (i.e., Web browser and Web server).
- All communication is stateless.
- The architecture is tiered, and data can be cached at any layer.

The main difference between RPC and REST is that REST uses the HTTP paradigm to directly address resources, while RPC provides a component-based interface with specialized methods.

[18] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm – requested in January 2011

3.2.1.2 Web Service Standards

Web service standards are used to ensure well-defined interfaces for language and platform neutral communication. They provide certain schemes which describe how data has to look like and how data has to be transmitted.

Simple Object Access Protocol

Simple Object Access Protocol (SOAP) is an XML-based messaging protocol for RPC-style operations. It is not tied to any particular operating system or programming language so theoretically the clients and servers in these dialogs can be running on any platform and written in any language as long as they can formulate and understand SOAP messages.

Every SOAP request consists of an envelope and a body as shown in listing 3:

```
<?xml version="1.0">

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Header>
        <!-- Header information -->
    </soap:Header>

    <soap:Body>
        <!-- Body Information -->
    </soap:Body>

</soap:Envelope>
```

Listing 3: SOAP Request Structure

Web Services Description Language

Web Services Description Language (WSDL) is an XML-based protocol used to describe web services, what public methods are available to them, and where the service is located [Hold08].

In contrast to SOAP which describes the data being transferred, WSDL describes the interface (known as the contract). Although SOAP holds type information, WSDL is more suitable for auto-generating language and platform independent proxies for web services. So, use the synergy of the two standards by defining documents with WSDL for transmission via SOAP.

Universal Discovery, Description, and Integration

UDDI is an open industry initiative, sponsored by the Organization for the Advancement of Structured Information Standards (OASIS), enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet[19].

UDDI, which is based on XML and SOAP, consists of two main parts: the specification for how to hold all of the information and the implementation of the specification.

3.2.2 Frameworks

A web application framework consists of reusable class libraries and design patterns to ease the implementation of frequently used functionality.

An important feature of frameworks is that they should work on a generic level so that they are suited for a multitude of applications. On the Web and the desktop, frameworks allow developers to concentrate on the application's requirements and on meeting deadlines, instead of on the mundane but necessary components that make applications run. [Hold08]

An application framework is not to be confused with an architectural framework, which is more like a meta-framework and placed on top of an application framework to impose more structure. Especially the Adobe Flex community pays a lot of attention to architectural frameworks. The three most popular are Cairngorm[20], Mate[21] and PureMV[22].

Application frameworks basically feature event and service maps and ensure that communication between UI and model is mediated by global objects (singletons). For instance, the view (UI) of an application is neither aware of the events/services to call in case of an action nor it knows which kind of data it needs or when data arrives. This allows a clear separation of development and maintenance work be-

[19] http://en.wikipedia.org/wiki/UDDI – requested in December 2010
[20] http://sourceforge.net/adobe/cairngorm/home/ – requested in February 2011
[21] http://mate.asfusion.com/ – requested in February 2011
[22] http://puremvc.org/ – requested in February 2011

tween designers and logic programmers. Discussing architectural frameworks in detail would go beyond the scope of this paper[23].

Many open, free available AJAX frameworks and a lot more JavaScript frameworks exist on the web. For an RIA developer it is important that they simplify the initialization of an AJAX request and DOM operations, provide a programming model suited for the MVC pattern and may offer high level GUI widgets.

Table 3 shows a selection of powerful, frequently used AJAX frameworks.

Framework	Short Summary
jQuery **http://jquery.com/**	• Minimalistic syntax for (challenging) operations • Good extension integration and thus countless plug-ins available • Provides with jQuery UI library for interactions, widgets and effects • Bundled with ASP.NET
MooTools **http://mootools.net/**	• Strong object orientation • Effects component, e.g. for transitions • Many functions for DOM manipulation
Prototype **http://www.prototypejs.org/**	• First JavaScript framework which introduced the `$()` function (also incorporated in jQuery) to substitute `getElementById()` and provides several other shortcuts • Uses `AJAX.Request()` for retrieving data and `AJAX.Updater()` for inserting it into the DOM
YUI (Yahoo UI Library) **http://developer.yahoo.com/yui/**	• Good API documentation • Holistic use of CSS selectors as element references • Cohesive set of widgets

Table 3: Selected AJAX Frameworks

[23] Please see [EDWF10] or http://www.adobe.com/devnet/flex/articles/flex_framework.html – requested in December 2010 – for more information on architectural frameworks.

3.3 Example Application

This section covers a demonstration of AJAX and its underlying technologies in practice. The application consumes two web services asynchronously via a Java servlet proxy, utilizes the jQuery framework and certain widgets from the jQuery UI library.

Application details:

- Consumes RESTful web services
- Implemented with jQuery framework
- Uses jQuery UI widgets to implement:
  - A loading animation during AJAX request
  - Auto completion of text field inputs
  - A pop-up calendar
- Developed with NetBeans IDE 6.9.1[24]
- Deployment on Glassfish 3 Server[25]

In jQuery the main object or global function is `$()`. It is mainly used to select document element from a specific context and replaces long function calls like `getElementById()`. `$(document).ready(function())` is similar to `window.onload()`, however it also waits for external content like advertisement.

Listing 4 illustrates the standard process for an AJAX request from `XMLHttpRequest` object creation, over request initiation to response parsing.

```
function initRequest() {
    xhr = createRequestObject();
    var from = document.getElementById("from");
    var to = document.getElementById("to");
    var url = "webserviceservlet?action=currency&from=" + from.value +
    "&to=" + to.value;
    xhr.open("GET", url, true);
    xhr.onreadystatechange = parseCurrencyResponse;
```

[24] http://netbeans.org/ – requested in December 2010
[25] http://glassfish.java.net/ – requested in December 2010

```
    xhr.send(null);
}
//Browser compatibility check
function createRequestObject() {
    if (window.XMLHttpRequest) {
      if (navigator.userAgent.indexOf('MSIE') != -1) {
          isIE = false;
      }
    return new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        return new ActiveXObject("Microsoft.XMLHTTP");
      }
}
function parseCurrencyResponse() {
    if ((xhr.readyState == 4) &&( xhr.status == 200)) {
      var response = xhr.responseXML;
      var curelement = response.getElementsByTagName("double")[0];
      var curvalue = curelement.childNodes[0].nodeValue;
      var curdiv = document.getElementById("curdiv");
      curdiv.appendChild(document.createTextNode(curvalue));
    }
}
```

Listing 4: Standard AJAX Processing

On a contrary to the standard AJAX processing from listing 4, listing 5 shows the obviously much simpler jQuery equivalent.

```
$.ajax({
    type: "GET",
    url: "webserviceservlet",
    data: "action=currency&from=" + from + "&to=" + to,
    //expected response datatype
    dataType: "XML",
    /*Callback function:
    Given to $.ajax() as parameter and executed after it.*/
    success: function(data){;
      //DOM manipulation
      $('#curdiv').append("1 " + from + " = " + data + " " + to);
    }
});
```

Listing 5: jQuery AJAX Processing

In order to provide a look & feel and behavior similar to desktop applications, AJAX applications can leverage the rich jQuery UI libraries.

For instance, the figures 14 to 16 visualize the ability of such plug-ins (not to be confused with runtime engine plug-ins[26]). For implementation details please see listing 6, which shows the whole script of the example application.

Figure 14 demonstrates the use of a loading circle plug-in to visualize the state of a request; loading bars also exist.

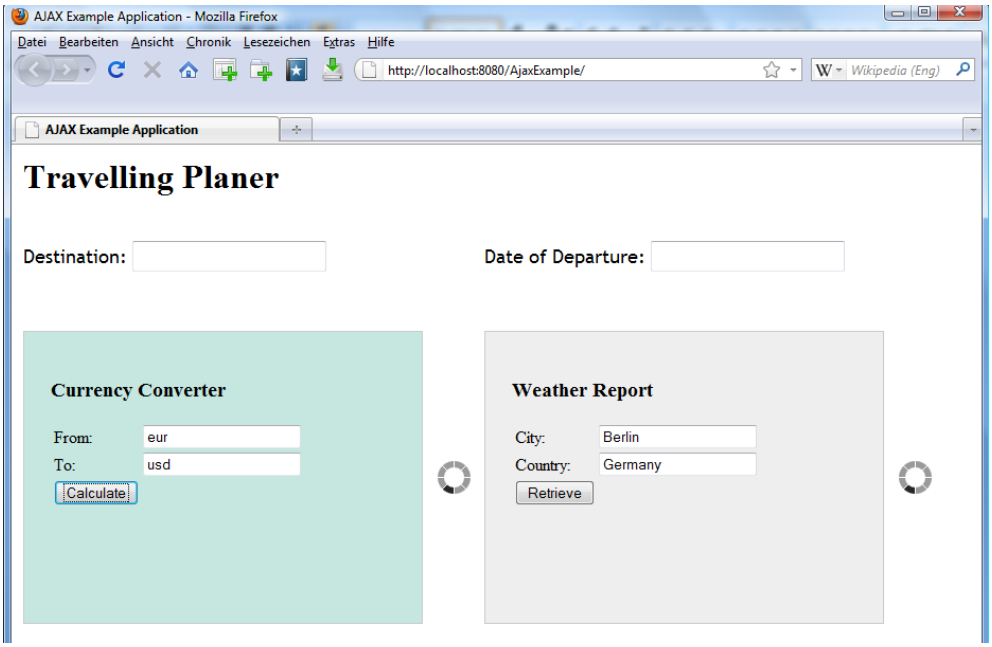


Figure 14: showLoading Widget in Action

[26] By definition they are basically the same; both add certain features to a host application. If the browser encounters references to content not subject to the core functionality of the browser, a third-party plug-in takes over the processing. E.g. a `.swf` file cannot be displayed by the browser; the Flash Player plug-in instead takes care of it. For a clearer separation, one may think of extensions when concerned with jQuery plug-ins and the like. They are small pieces of code working with existing functions to extend the usability of a specific program.

Auto completion like in figure 15 is one of the most frequently used features in modern web applications. The data source can be a local array of strings or a URL providing JSON data.

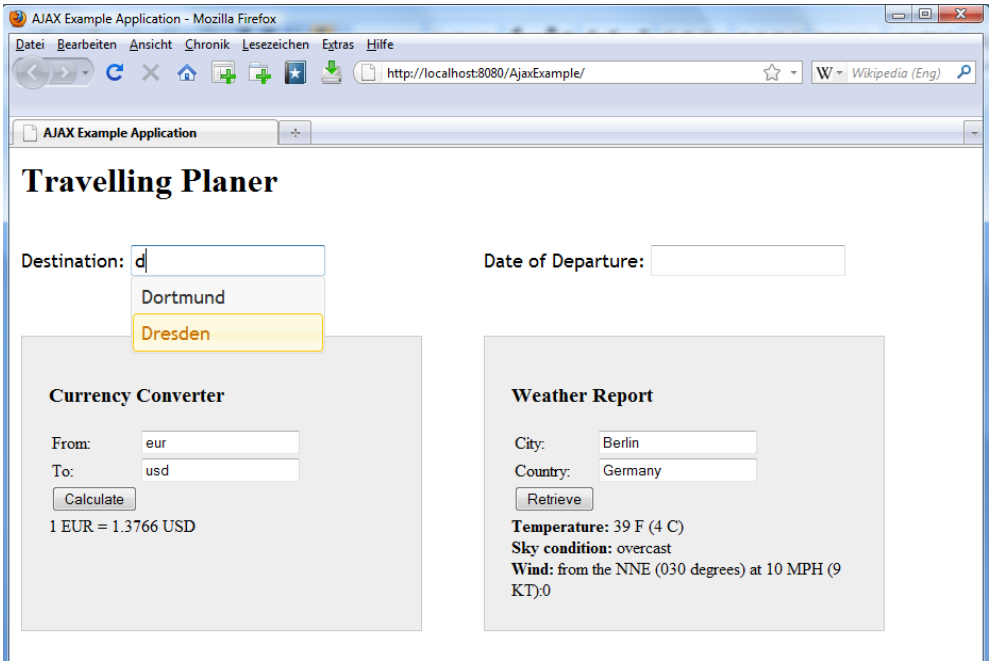


Figure 15: Auto Completion with jQuery UI Library

Finally, figure 16 shows a pop-up calendar which eases date input.

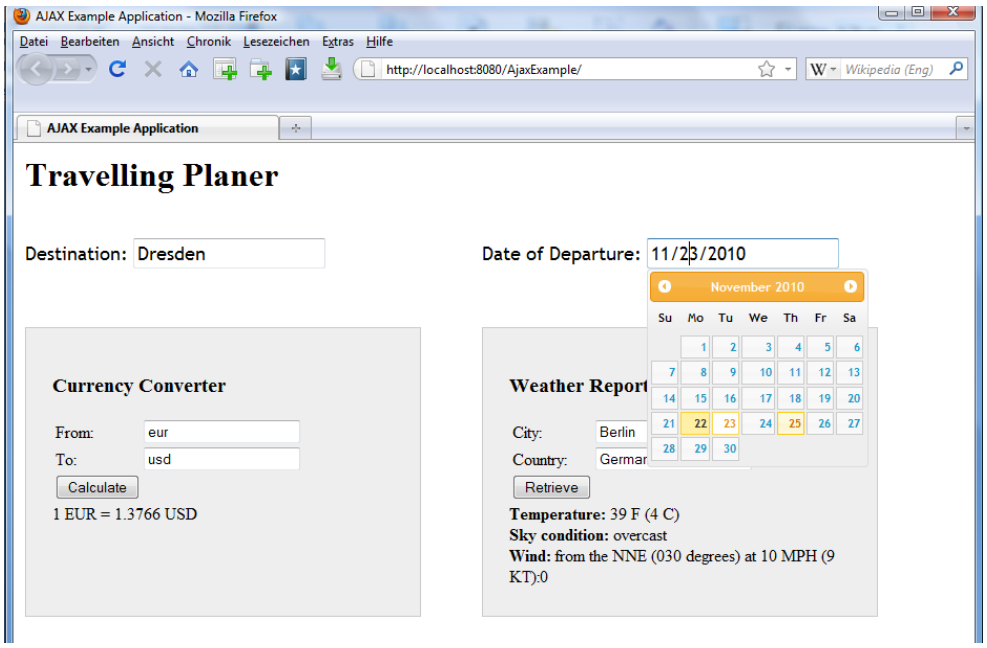


Figure 16: datePicker for Date Input

```
<script type="text/javascript">
    //jQuery similar to window.onload()
    $(document).ready(function (){
      //binds a new autocomplete widget to the assigned HTML element
      $("#auto").autocomplete({
        source: ["Berlin", "Bonn", "Dortmund", "Dresden", "Frankfurt",
                 "Hamburg"]
      });
      //binds a new datepicker widget to the assigned HTML element
      $("#date").datepicker({clickInput: true});
      //changes the style class of the element dynamically on focus
      $("div.wsout").focusin(function(){
        $(this).removeClass().addClass("wsin");
      }).focusout(function(){
        $(this).removeClass().addClass("wsout");
      });
      //binds a click event listener to the assigned HTML element
      $('#curbutton').click(function (event) {
        var from = $('#from').val().toUpperCase();
        var to = $('#to').val().toUpperCase();
        //loading animation
        $('#loadingcur').showLoading();
        //jQuery AJAX function
        $.ajax({
          type: "GET",
          url: "webserviceservlet",
          data: "action=currency&from=" + from + "&to=" + to,
          dataType: "XML",
          /*Callback function:
          Given to $.ajax() as parameter and executed after it. */
          success: function(data){;
            $('#loadingcur').hideLoading();
            //DOM manipulation
            $('#curdiv').append("1 " + from + " = " + data + " " + to);
          }
        });
        event.preventDefault();
      });
      //binds a click event listener to the assigned HTML element
      $('#weatherbutton').click(function (event) {
        //loading animation
        $('#loadingwea').showLoading();
        //jQuery AJAX function
        $.ajax({
          type: "GET",
          url: "webserviceservlet",
          data: "action=weather&city=" + $('#city').val() + "&country=" +
          $('#country').val(),
          dataType: "XML",
```

```
          /*Callback function:
          Given to $.ajax() as parameter and executed after it.*/
          success: function(xml){
            $('#loadingwea').hideLoading();
            //DOM manipulation
            var temp = $(xml).find('Temperature').text();
            var sky = $(xml).find('SkyConditions').text();
            var wind = $(xml).find('Wind').text();
            $('#weatherdiv').append("<b>Temperature: </b>" + temp + "<br>");
            $('#weatherdiv').append("<b>Sky condition: </b>" + sky + "<br>");
            $('#weatherdiv').append("<b>Wind: </b>" + wind);
          }
        });
        event.preventDefault();
      });
    });
</script>
```

Listing 6: AJAX Client Application

Communication with the two RESTful web services is mediated by a servlet, because the script itself is only able to call services from its own source domain. To achieve remote web service invocation, the `doGet()` method of the servlet has to be overwritten to resolve parameters from the initial request and to contact the service.

The response from the web service is formatted as necessary to syntactically correct XML and forwarded to the client as in listing 7.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      StringBuffer sb = new StringBuffer();
      String urlstring = null;
      String action = request.getParameter("action");

      try {
        if (action.equals("currency")) {
          String from = request.getParameter("from");
          String to = request.getParameter("to");
          urlstring = "http://www.webservicex.net/CurrencyConvertor.asmx/" +
                  "ConversionRate?FromCurrency=" + from + "&ToCurrency=" + to;
        } else {
          String city = request.getParameter("city");
          String country = request.getParameter("country");
```

```
            urlstring = "http://www.webservicex.net/globalweather.asmx/" +
                        "GetWeather?CityName=" + city + "&CountryName=" + country;
          }

        //Resource calling and reading on remote URL.
        URL url = new URL(urlstring);
        URLConnection urlc = url.openConnection();
        urlc.connect();
        BufferedReader in = new BufferedReader(new InputStreamReader(
          urlc.getInputStream()));
          String inputLine;
          String outputLine;

          //Response formatting, necessary for faultless DOM parsing.
          if (action.equals("weather")) {
            int i = 0;
            while ((inputLine = in.readLine()) != null) {
              i++;
              if(i==2){continue;}
              outputLine = inputLine.replaceAll("&lt;", "<").replaceAll("&gt;",
              ">").replaceAll("</string>", "");
              sb.append(outputLine);
            }
          } else {
            while ((inputLine = in.readLine()) != null) {
              sb.append(inputLine);
            }
          }
          in.close();
          response.setContentType("text/xml");
          response.setHeader("Cache-Control", "no-cache");
          //Sends web service response to initial requester.
          out.write(sb.toString());
        } finally {
          out.close();
        }
}
```

Listing 7: doGet() Method of the Java Servlet

4 Adobe Flex

The Flex ecosystem is a set of libraries, tools, languages and deployment runtimes that provides an end-to-end framework for designing, developing and deploying RIAs.[27] Flex should not be confused with Flash. Flash is the name of the platform Flex applications are developed for. So Flex has integrated support from other Flash products. For instance, the UI architect can create the view with special graphic software like Adobe Catalyst while the programmer is coding the application logic with Flash Builder as they share the same project format. Sometimes Flash also refers to the authoring tool used to develop animation and multimedia content. While the Flash authoring tool is suited for design tasks, the Flex API provides rich functionality for both user interface creation and application logic development.

4.1 Specification

Since version 3 released on February 25 2008, the Flex SDK is available under open-source license. The primary IDE for Flex applications is the Flash Builder[28]. It is based on the Eclipse IDE, features a WYSIWYG-Editor and allows easily connecting your application to services. The Flash Builder is free of charge for students and unemployed developers. A good free, open source alternative is FlashDevelop[29].

4.1.1 Runtime Environments

Flash applications are executed in three different runtime environments: Flash Player, Adobe Integrated Runtime (AIR) and Flash Lite. The Flash Player is used to embed Flash content in a web page while AIR runs installed applications on the desktop. Flash Lite is intended for the mobile platform and features a subset of the Flex API. The latest version 4, released on March 22 2010, supports ActionScript 3 and is integrated in Symbian^3[30].

---

[27] http://flextutorial.org/2009/07/22/flex-101-with-flash-builder-4-part-1/ – requested in December 2010
[28] http://www.adobe.com/products/flashbuilder/ – requested in December 2010
[29] http://www.flashdevelop.org/wikidocs/index.php?title=Main_Page – requested in December 2010
[30] http://wiki.forum.nokia.com/index.php/Flash_Lite_4_in_Nokia_Symbian%5E3_devices – requested in December 2010

4.1.2 Language Characteristics

Flex offers great freedom concerning the way an application can be coded. Basically all work can be done with the object-oriented scripting language ActionScript (AS) but also with the declarative XML-based MXML due to the fact that MXML code is compiled to AS code.

ActionScript 3

Since release version 3 released on June 27 2006, ActionScript (AS) is a strongly typed object-oriented language with similarities to Java and C#. It is a dialect of the ECMAScript scripting language standard as JavaScript is.

The key features of AS 3 are [Moock07]:

- First-class support for common object-oriented constructs, such as classes, objects, and interfaces
- Single-threaded execution model
- Runtime type-checking
- Optional compile-time type-checking
- Dynamic features such as runtime creation of new constructor functions and variables. Instances from classes with the attribute “dynamic” can have instance variables added at runtime
- Runtime exceptions
- Prototyping:
  Use the prototype object of a class to share information and behavior among all instances of the class and instances of descendents.
- Direct support for XML as a built-in data type
- Packages for organizing code libraries
- Namespaces for qualifying identifiers
- Regular expressions

Listing 8 shows the structure of an AS class creating graphical elements. In ActionScript 3 all graphical objects inherit from the `DisplayObject` class. Every descendent from `DisplayObject` can be positioned, rotated and sized. To visualize components on screen they have to be added to the display list.

The display list is the hierarchy of all graphical objects currently displayed by the Flash runtime. When a display object is added to the display list and is positioned in a visible area, the Flash runtime renders the content of that display object to the screen [Moock07].

The display list is either represented by `Sprite` or `MovieClip`. By invoking the `addChild` method of the Sprite class the label objects gets visualized.

```
package
{
	import flash.display.*;
	import flash.text.*;

	public class Main extends Sprite
	{
		private var label:TextField;

		public function Main()
		{
			label = new TextField();
			label.text = "Some Text";
			label.x = 100;
			label.y = 50;
			label.selectable = false;

			addChild(label);
		}
	}
}
```

Listing 8: ActionScript 3 Display Example

MXML

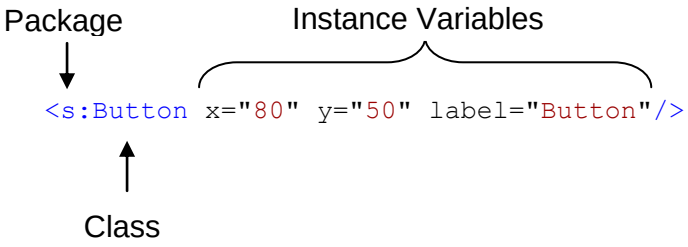
MXML[31] is an XML-based language with a declarative syntax designed for user interface related tasks. Olof Torgersson, an analyst of declarative languages characterizes them as following: "*From a programmer's point of view, the basic property is that programming is lifted to a higher level of abstraction. At this higher level of ab-*

[31] Adobe gives no official meaning for the acronym MXML, but it probably is derived from the MX namespace Macromedia introduced and consequently stands for "Macromedia eXtensible Markup Language".

straction the programmer can concentrate on stating what is to be computed, not necessarily how it is to be computed"[32].

Web developers may find MXML more appealing than ActionScript due to its syntax similar to that of languages they are already familiar to.

MXML tags have the following structure:



When a tag is inserted, an instance of the appropriate class is created in the background in ActionScript. Listing 9 and listing 10 contrast the creation of a button and wiring it up with some event listener in MXML and in AS. It is necessary to write the event handler function in a `<fx:Script>` tag, though. The `init()` method in listing 10 gets called on application startup.

Flex encourages the use of MXML because it has a more human readable syntax than AS and fewer code lines are needed for certain tasks.

```
<s:Application>
  <fx:Script>
       <![CDATA[
         import mx.controls.Alert;

         protected function buttonClick(event:MouseEvent):void
           {
                Alert.show("Button Clicked!");
             }
       ]]>
  </fx:Script>
  <s:Button x="79" y="64" label="Button" click="buttonClick(event)"/>
</s:Application>
```

Listing 9: Button Creation with MXML

[32] Torgersson, Olof. "A Note on Declarative Programming Paradigms and the Future of Definitional Programming," Chalmers University of Technology and Göteborg University, Göteborg, Sweden. http://www.cs.chalmers.se/~oloft/Papers/wm96/wm96.html – requested in December 2010

```
<s:Application initialize="init()">
  <fx:Script>
    <![CDATA[
      import mx.controls.Alert;
      import mx.controls.Button;

      private var button:Button;
      private function init():void
        {
          button = new Button();
          button.label = "Click Me!";
          button.addEventListener(MouseEvent.CLICK, buttonClick);
          addElement(button);
        }
      private function buttonClick(e:MouseEvent):void
        {
          Alert.show(e.currentTarget.toString());
        }
     ]]>
  </fx:Script>
</s:Application>
```

Listing 10: Button Creation with ActionScript 3

4.1.3 Deployment

Human-written AS code is compiled to AS bytecode. This bytecode is wrapped in a binary container as a `.swf` file. The `.swf` file ending refers to the standard format of every Flash application for execution on a Flash runtime. The only exceptions are applications destined for the desktop which have the `.air` format.

Although `.swf` files can run in the standalone mode of the Flash Player it is more common to embed them with the `<embed>` (Netscape-based browsers) or `<object>` (Internet Explorer) HTML tag for replay with the browser plug-in Flash Player in a web page. At first write a JavaScript file that creates the specific tags. Simply apply the `<embed>` tag nested within the `<object>` to ensure holistic compatibility with all modern browsers (cf. listing 11).

```
<!-- JsWrapper.js -->
document.write("<object id='FlashApp'
    classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
    codebase='http://download.macromedia.com/pub/
    shockwave/cabs/flash/swflash.cab#version=9,0,0,0'
    height='200' width='300'>");
document.write("<param name='movie' value='JSInteract.swf'/>");
document.write("<embed name='FlashApp' src='JSInteract.swf'
    plugins-page='http://www.macromedia.com/shockwave/' +
```

```
        'download/index.cgi?P1_Prod_Version=ShockwaveFlash'
        height='200' width='300'/>");
document.write("</object>");
```

Listing 11: Embed Flash Content with JavaScript

Subsequently, apply the JavaScript wrapper to a HTML page as in listing 12:

```
<!-- page.html -->
<html>
  <body>
    <script src="JsWrapper.js"></script>
  </body>
</html>
```

Listing 12: JavaScript Wrapper for Flash Content Applied to HTML

4.2 Data Binding and Data Validating

One of the most crucial elements of RIA development is the principle of data binding. Data binding mechanisms allow synchronizing the value/state of one object (property) to another at runtime without cluttering up your code with complex change listening scripts or getter and setter methods. Another tool simplifying frequently used routines is data validating. It allows checking input data on the client side which results in a more responsive application and in a decrease in server load.

Data Binding

Data binding can be accomplished by assigning the value of a property of object X as value of a property of object Y. As an example, depicted in listing 13, the `text` property of `<s:TextInput>` with the id “textfield” will automatically be assigned to the `text` property of `<s:Label>` and updated every time the `text` property of “textfield” changes. (Note that the `text` property of “textfield” is initially empty and blanked until the used makes some input.) Bound data is recognized via curly braces surrounding the bound property.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application  xmlns:s="library://ns.adobe.com/flex/spark"
                minWidth="955" minHeight="600">
       <s:Label x="74" y="52" text="{textfield.text}"/>
       <s:TextInput id="textfield" x="124" y="52"/>
</s:Application>
```

Listing 13: Flex Object Value Data Binding

Another option is to bind a whole variable. Such a “bindable” variable has to be declared along with the `[Bindable]` metadata tag. Listing 14 demonstrates how an array collection containing feed data is bound to a data grid as data provider. The private variable `feed` of type `ArrayCollection` is assigned as value to the `dataProvider` property of `<mx:DataGrid>`, surrounded by curly braces signalizing data binding. As a consequence, every time the feed is updated the data grid also reflects the new data.

```
[Bindable]
private var feed:ArrayCollection;
<mx:DataGrid x="10" y="40" width="400" height="125" dataProvider="{feed}">
 <mx:columns>
       <mx:DataGridColumn headerText="Title" dataField="title"/>
       <mx:DataGridColumn headerText="Date" dataField="date"/>
 </mx:columns>
</mx:DataGrid>
```

Listing 14: Flex Metadata Tag Data Binding

Data Validating

Traditionally, client input data had to be sent to the server for validation and error feedback was returned if the data was not suitable for processing. The `mx` package contains a wealth of data formatters and validators.

For example an `EmailValidator` as illustrated in figure 17 looks for the existence of an @ sign and a correct domain.

```
<mx:EmailValidator id="eVal" valid="eVal_valid(event);"
                             invalid="eVal_invalid(event);"
```

Figure 17: Flex Email Validating

When the “Check” button is pressed the `validate()` function of the validator is called to verify the text input. In case of valid input (email address is a string with a single @ sign and a period in the domain) the function specified at the `valid` property of the `<mx:EmailValidator>` is called, otherwise a certain function from the `invalid` property is invoked. In the example from figure 17 the error message the validation result returns is assigned to the `errorString` property of the input field, in case of invalid input.

4.3 Remote Service Handling

Analog to the `XMLHttpRequest` object from AJAX, discussed in section 3.1.4, Flex provides functionality to asynchronously send data to a remote server and to receive data. In addition, the Flex SDK leverages a specialized language for more comfortable XML data parsing than with standard DOM. Once again, the communication with the server can be accomplished both with ActionScript and MXML.

As demonstrated in listing 15, the very first step is to create an `URLRequest` object with the URL to connect to. Next is the `URLLoader` object, which actually executes the request. Finally the `URLLoader` object is wired up with an event listener and a callback method before invoking the `load()` method on `urlLoader` with the passed `urlRequest`.

```
private function getFeed():void {
  var url:String = "http://localhost/feed.xml";
  var urlRequest:URLRequest = new URLRequest(url);
  var urlLoader:URLLoader = new URLLoader();
  urlLoader.addEventListener(Event.COMPLETE, feedLoaded);
  urlLoader.load(urlRequest);
}
```

Listing 15: Data Loading with ActionScript 3

The `HTTPService` tag in listing 16 is the MXML equivalent to the `urlLoader` AS approach from listing 15. It requires a method (`GET` or `POST`) and a callback function to be specified. Use `httpService.send()` to execute the request formulated with `<mx:HTTPService>` and ultimately to load data similar to executing `urlLoader.load(urlRequest)`.

```
<mx:HTTPService
  id="httpService"
  url="http://localhost/feed.xml"
  method="GET"
  result="feedLoaded" />
```

Listing 16: Data Loading with MXML

ActionScript 3.0 implements ECMAScript for XML (“E4X”), an official ECMA-262 language extension for working with XML as a native data type. E4X seeks to improve the usability and flexibility of working with XML in ECMA-262-based languages (including ActionScript and JavaScript). [Woock07]

In previous versions of ActionScript, one had to use `childNodes[]` and actually drill down into the XML object in order to find the information one was searching for. Now in ActionScript 3, one can simply say `xml..itemToBeFound`, and it automatically finds that correct item. [KeCh09] The descendant operator (..) returns an XMLList representing all descendants (not just child nodes but also nodes at any level lower than the addressed element at the XML hierarchy) that match the identifier “itemToBeFound”. If there were multiple items, you would build a loop to go through them.

As demonstrated in listing 17, the title, date of publication and the message properties of each item in the received feed are parsed and added to an `ArrayCollection`. Such an `ArrayCollection` can be set as the `dataProvider` property of a `DataGrid` or the like to present the data in a structured way (cf. the example in section 4.3.1).

```
private function feedLoaded(e:Event):void {
  var urlLoader:URLLoader = URLLoader(e.target);
  var xml:XML = XML(urlLoader.data);
  feedTitle = String(xml..title[0]);
  feed = new ArrayCollection();
  for each(var item:* in xml..item)
  {
     feed.addItem({
       title:item..title,
       date:item..pubDate,
       content:item..message
    });
  }
}
```

Listing 17: Handling Response Data with E4X

Please note the curly braces ({}) surrounding the variables passed on invocation of the `addItem()` function of the "feed" `ArrayCollection`. The `addItem()` function basically only takes one argument. In order to pass multiple variables however, the variables have to be passed as block statement (sometimes referred to as "stem variable") enclosed by a pair of curly braces similar to the body of a method or of a class.

4.3.1 Data Service Wizards

Integrating with an external service can be a time consuming, error prone process. You have to be aware of the data type and the structure of the response, parse it accordingly and make use of the returned values in your application. The Flash Builder – the primary IDE for Flex applications – offers wizards which ease the integration of external services by auto-generating related client code as long as those services return XML or JSON data. They auto-detect the return type of the service by sending a dummy request and encapsulate the returned values as properties of a class in the background. A user-named operation (function) gives access to these properties.

The following basic steps are necessary to integrate an HTTP (RESTful) service[33] with a Flash Builder project consuming industry news RSS feeds from `http://finance.yahoo.com/rss/industry` via wizard:

- Select `Data -> Connect to Data/Service` and choose `HTTP`
- Configure the HTTP service (cf. figure 18):

  Name the operation responsible for invoking the service, select the method (GET or POST) and insert the URL of the service.

  Parameters: Define the parameters attached to the request. The number and names of the parameters must match the API specification.

[33] Please see http://help.adobe.com/en_US/Flex/4.0/AccessingData/WSbde04e3d3e6474c4-668f02f4120d422cf08-7ffe.html – requested in December 2010 – for a list of all kinds of services Flash Builder is able to connect to and how to access them.

Service Name: Give the service a name later on referred to.

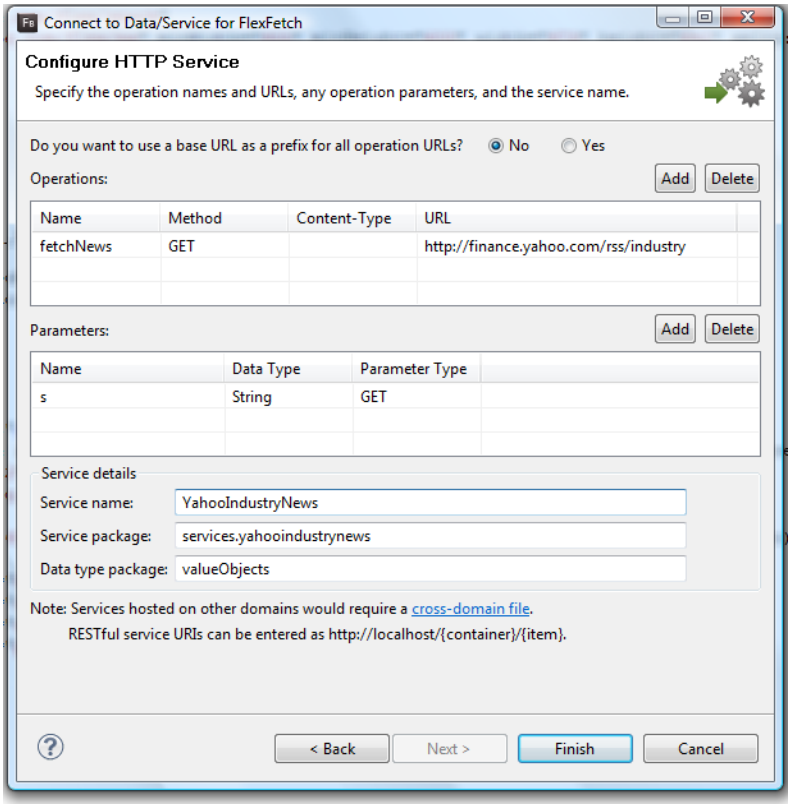


Figure 18: HTTP Service Configuration

- Bind results and configure the return type:

  Assign a return value object to the `dataProvider` property of a structuring display component like a `dataGrid`.

  Enter valid parameters and call the operation for a dummy request. The response data type is auto-detected according to the returned values. In case of XML, Flash Builder maps the response to a tree-like structure as illustrated in figure 19.

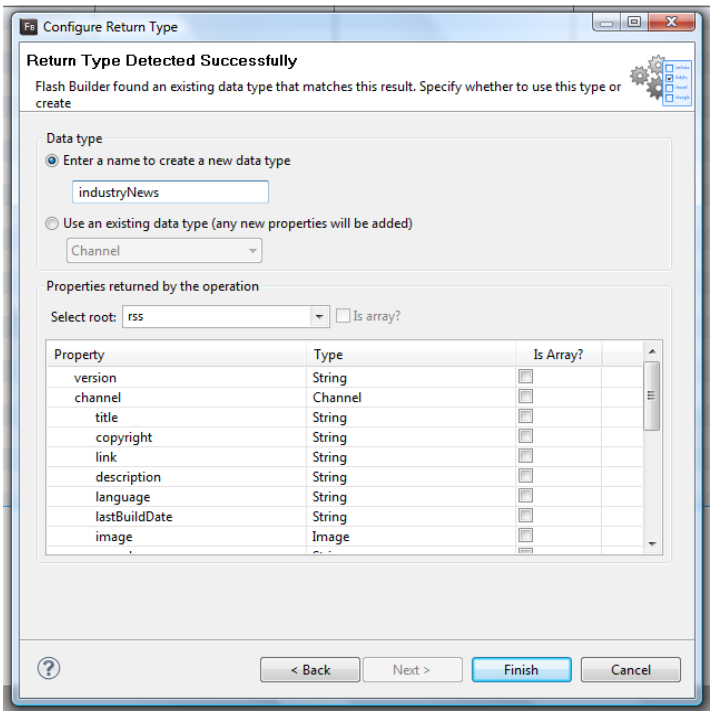


Figure 19: Preview on Service Response Properties

Non-leaf nodes are marked as arrays. All properties of the selected root property are displayed as columns in the `dataGrid`. All properties of the response can be manually and arbitrarily assigned using the basic property hierarchy notation.

Listing 18 demonstrates the source code of a simple example which calls the encapsulated service and displays the results. The `lastResult` property of the `getNewsResult` object stores the last successfully fetched result.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                    xmlns:s="library://ns.adobe.com/flex/spark"
                    xmlns:mx="library://ns.adobe.com/flex/mx"
                    minWidth="955" minHeight="600"
                    xmlns:financenews="services.financenews.*">
      <fx:Script>
            <![CDATA[
                    import mx.controls.Alert;
                    import mx.events.FlexEvent;

                    protected function fetchNews(event:MouseEvent):void
                    {
          getNewsResult.token = financeNews.getNews(searchTerms.text);
                    }

            ]]>
```

```
        </fx:Script>
        <fx:Declarations>
               <s:CallResponder id="getNewsResult"/>
               <financenews:FinanceNews id="financeNews"
                   fault="Alert.show(event.fault.faultString + '\n' +
                   event.fault.faultDetail)" showBusyCursor="true"/>
        </fx:Declarations>
        <s:Label x="65" y="36" text="Search Terms:"/>
        <s:TextInput x="149.55" y="26.1" id="searchTerms"/>
        <s:Button x="294.65" y="26.45" label="Fetch News"
                  click="fetchNews(event)"/>
        <mx:DataGrid x="65" y="105" id="dataGrid"
                  dataProvider="{getNewsResult.lastResult.channel.item}"
                  width="70%" height="50%">
               <mx:columns>
                     <mx:DataGridColumn headerText="Title"
                       dataField="title" itemRenderer="NewsRenderer"/>
                     <mx:DataGridColumn headerText="Link"
                       dataField="link"/>
                     <mx:DataGridColumn headerText="Publication Date"
                       dataField="pubDate"/>
               </mx:columns>
        </mx:DataGrid>
        <s:Label x="65" y="85"
          text="{getNewsResult.lastResult.channel.title}"/>
</s:Application>
```

Listing 18: Connecting to HTTP Service via Flex Wizard

When the user has entered search terms and clicks on “Fetch News”, the `dataGrid` is filled and the `title` property of the `getNewsResult.lastResult.channel` object is displayed as headline (cf. figure 20).

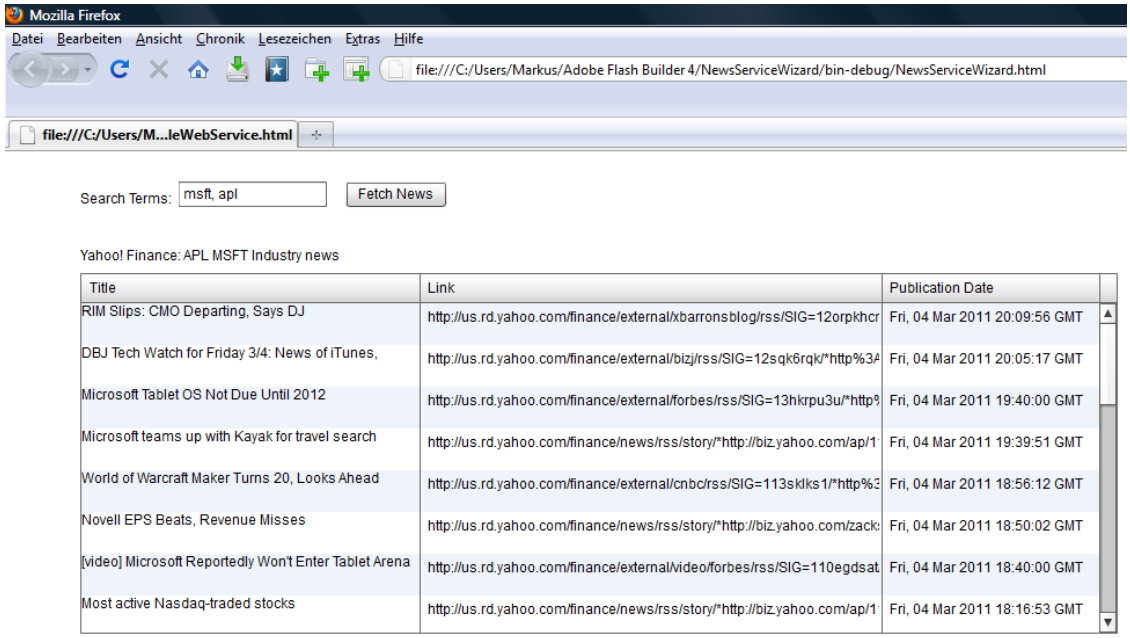


Figure 20: Flex Data Service Example

4.3.2 AMF

The protocols used for data loading from a remote source and sending data to it are text-based. So, the received data has to be manually parsed and typed for further processing. To avoid these time-consuming tasks, Flex employs the open Action Message Format (AMF), which ensures receiving strongly typed data.

AMF is a compact binary format that is used to serialize ActionScript objects into a sequence of bytes of the class `ByteArray`, which contains all required information about the structure of the original object. Because AMF's format is open to all, Adobe as well as third-party developers can implement it in various products to deserialize such pieces of binary data into an object in a different VM (Virtual Machine), which does not have to be Flash Player. [EDWF10]

Placing the `[RemoteClass]` metadata tag before a class declaration as in listing 19 ensures the "conservation" of a data type:

```
package
{
    [RemoteClass(alias="com.MyAsDataTypeClass")]
    public class MyAsDataTypeClass{
    }
}
```

Listing 19: RemoteClass for AMF-based Transmition

4.4 Interaction with the Environment

In large enterprises, usually you don't start a new Enterprise Flex project from scratch without worrying about existing web applications written in JSP, ASP, AJAX, and the like. More often, enterprise architects gradually introduce Flex into the existing web fabric of their organizations. Often, they start with adding a new Flex widget into an existing web page written in HTML and JavaScript, and they need to establish interaction between JavaScript and ActionScript code. [EDWF10]

The Flex SDK offers three approaches for setting up communication between JavaScript and AS: `ExternalInterface`, Flex AJAX Bridge and the `flashVars` variable.

ExternalInterface

The `ExternalInterface` class allows mapping ActionScript and JavaScript functions. Before ActionScript functions can be invoked from JavaScript one has to add a callback:

```
ExternalInterface.addCallback("sendDataToFlash", getDataFromJavaScript);
```

When JavaScript invokes the `sendDataToFlash()` functions the specified callback method is invoked. Calling a JavaScript function from within your Flash application is accomplished with:

```
var result:Object = ExternalInterface.call("getDataFromFlash", jsArgument);
```

Listing 20 and listing 21 demonstrate the `ExternalInterface` principle. The `sendText()` function sends the input text to the external environment by calling the `getDataFromFlash()` function declared in JavaScript.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/mx"
                      minWidth="955" minHeight="600"
                       >
       <s:Button x="187" y="66" label="Send" id="send_button"
          click="sendText(event)"/>
       <s:TextInput x="37" y="65" id="sending_ti" enabled="true"/>
       <s:Label x="37" y="48" text="Sending to JavaScript" width="122"
          height="22"/>

       <fx:Script>
              <![CDATA[
                     import flash.events.Event;
                     import flash.external.ExternalInterface;

                     public function sendText(event:MouseEvent):void
                     {
                            var jsArgument:String = sending_ti.text;
                            var result:Object =
                              ExternalInterface.call("getDataFromFlash",
                              jsArgument);
                     }
              ]]>
       </fx:Script>
</s:Application>
```

Listing 20: Flash Application Calling JavaScript Function

```
<html>
  <head>
    <script>
      function getDataFromFlash(str) {
        document.myForm.receivedField.value = "From Flash: " + str;
      }
    </script>
  </head>

  <body>
   <embed src='JSInteract.swf' height='100' width='550' id='EISample'>
   <div style="margin-left:35px; margin-top:35px;">
     <form name="myForm">
       <p>Received from ActionScript</p><br>
       <input type="text" name="receivedField">
     </form>
   <div>
  </body>
</html>
```

Listing 21: HTML File for Interaction with ActionScript

JavaScript receives the passed string and displays it in a form (cf. figure 21).

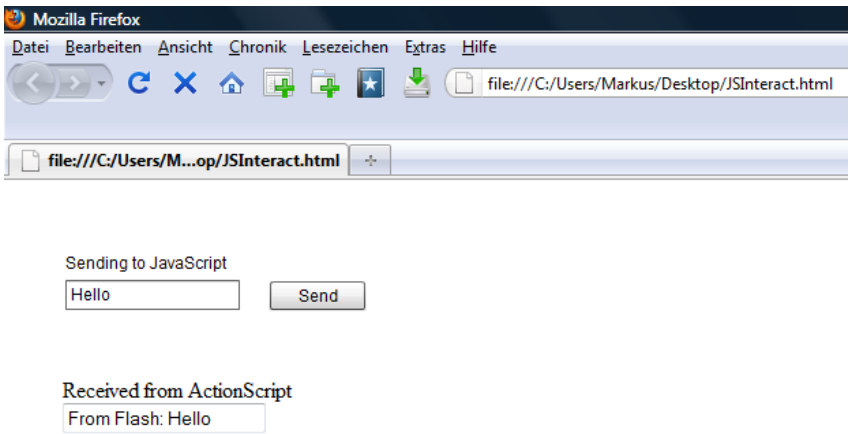


Figure 21: ExternalInterface Example

ActionScript may block that kind of interaction, depending on the applied security policy. ActionScript assigns a security status known as a security-type to every `.swf` file opened by or loaded into Flash Player. There are four possible security-types: `remote`, `local-with-filesystem`, `local-with-networking`, and `local-trusted`.

Each security-type defines a distinct set of rules that governs a `.swf` file's ability to perform external operations. Specifically, the types of external operations a security- type can potentially prohibit include [Moock07]:

- Loading content
- Accessing content as data
- Cross-scripting
- Loading data
- Connecting to a socket
- Sending data to an external URL
- Accessing the user's camera and microphone
- Accessing local shared objects
- Uploading or downloading files selected by the user
- Scripting an HTML page from a `.swf` file and vice versa
- Connecting to a `LocalConnection` channel

ActionScript assigns the security-type depending on the location from which the application was loaded and the privileges given to the `.swf` file. What is more, a `.swf` file can be explicitly trusted by granting it local trust[34]. Nevertheless, the resource distributor can set up an own policy file. A policy file gives `.swf` files from its list of trusted origins access to resources that would otherwise be inaccessible due to Flash Player's security restrictions (cf. listing 22.)

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy
  SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
    <cross-domain-policy>
      <allow-access-from domain="example1.com"/>
      <allow-access-from domain="example2.com"/>
      <allow-access-from domain="example3.com"/>
    </cross-domain-policy>
```

Listing 22: Flash Policy File

[34] http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html – requested in December 2010

Flex AJAX Bridge

With the Flex AJAX Bridge one can make certain ActionScript classes available to JavaScript without any additional coding. In contrast to the `ExternalInterface`, it is not necessary to write an extra library in which the accessible functionality is specified. The following code line gives access to the application instance from JavaScript:

```
var flexApp = FABridge.flash.root();
```

Use this instance to call any function declared in the ActionScript file, but also to pass functions (e.g. event handlers).

The flashVars Variable

While embedding a `.swf` file in HTML, Flash Builder includes `flashVars` parameters in the tags `<Object>` and `<Embed>`. ActionScript code can read them using `Application.application.parameters` [EDWF10].

An auto-generated HTML file contains the JavaScript variable `flashVars`. Include parameters in the form of key/value pairs separated by “&”. A “+” sign represents a single blank space. ActionScript can work with such parameters as they were ordinary variables. For example when using an `<object>` tag for integration of a `.swf` file write

```
<param name=FlashVars value="myNameAsFlashVar=John+Doe">
```

or when using an `<embed>` tag write

```
<embed href="test.swf" FlashVars="myNameAsFlashVar=John+Doe" ></embed>
```

to include a parameter. In ActionScript subsequently read and assign the set parameter to a variable:

```
var myName:String = Application.application.parameters.myNameAsFlashVar;
```

4.5 Adobe Integrated Runtime (AIR)

An AIR application is a downloadable gadget installed like a standard desktop application, which means it does not run in the sandbox of the browser. Hence, an AIR application is free from security and usability restrictions that limit Flash Player based RIA applications.

The following comparison in table 4 points out the main differences between a browser RIA and an AIR RIA.

Feature	In the Browser	Standalone Application
Installation	Only Runtime Environment (Flash Player) installed.	Install Runtime Environment (AIR) and application.
Application updates	Applications are updated by pushing new content to a website.	Install new version of application
Programming languages	JavaScript is provided by browsers, and ActionScript is provided by Flash Player software.	Integrated JavaScript and ActionScript virtual machines are compatible with the browser.
Background capability	Only in a visible browser window.	Run in the background or provide notifications.
Persistence	Activity is limited to the browser session. When the browser is closed, information is lost.	Store information locally and operate offline.
System integration	Applications are sandboxed, so device integration is limited.	Applications can access a local file system and device APIs.
Data storage	Limited local storage, which the browser can destroy.	Unlimited local storage and access to a local database, plus encrypted local storage.

Table 4: Flash Player vs. AIR Application [AIRComp]

AIR applications can leverage native menus and the system clipboard. Since version 2.0, released on November 16 2009, it is also possible to launch and communicate with native (non-AIR) applications, detect mass storage devices and open files with default programs.

Although Flex does not have language elements or libraries that can work with a relational DBMS, AIR comes bundled with a version of SQLite that is installed on the client and is used to create a local database (a.k.a. local cache) to store application data in the disconnected mode [EDWF10].

Imagine a sales representative visiting clients based on daily routes dispatched by a central corporate database. The agent's work does not require any tools except the application. However, if the connection to the remote server is cut off or the agent has/wishes to continue working offline, relevant data is stored to the local database. Next time an available network is detected – AIR automatically can detect networks – the local database is synchronized with the one of the server.

An AIR application also look like a native window of the underlying OS, due to the simple fact, that is uses the `<s:WindowedApplication>` root tag instead of `<s:Application>`.

The class `flash.filesystem.File` is a means of getting access to the files and directories on the user's computer. This class allows creating, moving, copy, or deleting files. For read/write operations, use the class `FileStream` from the package `flash.filesystem` [EDWF 10].

The process of writing files, for instance, is very similar to, let's say, the Java one:

- Create a stream instance:

```
var myFileStream4Write: FileStream = new FileStream();
```

- Open the stream in writing mode with specifying the file :

```
myFileStream4Write.openAsync(myFile,FileMode.WRITE);
```

- Write a byte string to the file using a character set:

```
myFileStream4Write.writeMultiByte(textToWrite, "iso-8859-1");
```

- Close the stream:

```
myFileStream4Write.close();
```

The AIR API features some additional controls for file system navigation like the `<mx:FileSystemTree/>` tag shown in figure 22.

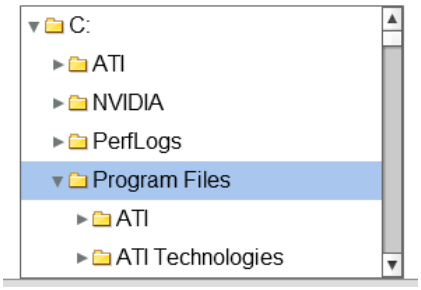


Figure 22: AIR FileSystemTree Control

Once your application is ready for deployment the `.air` distribution package can be created. This file, which is based on `.zip` compression, contains all data needed to install AIR applications. But first it has to be signed with a certificate, since AIR applications enjoy access privileges to the local system. Figure 23 demonstrates the assignment of a certificate. Select an existing certificate or create a new one with a password.

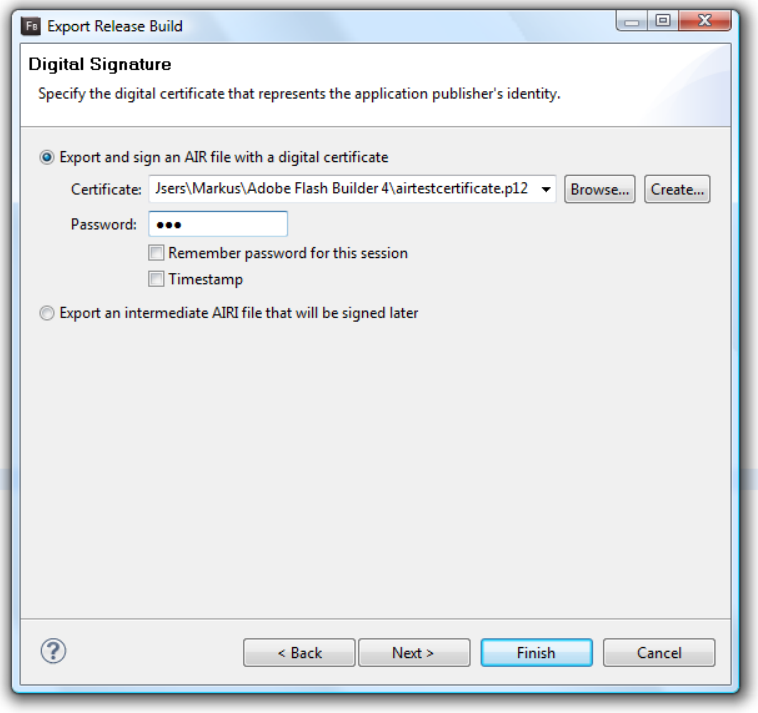


Figure 23: Sign AIR application

Click the `.air` file for prompt installation (cf. figure 24). Unless a real digital certificate is used, the AIR installer warns the user about the unknown identity of the publisher.

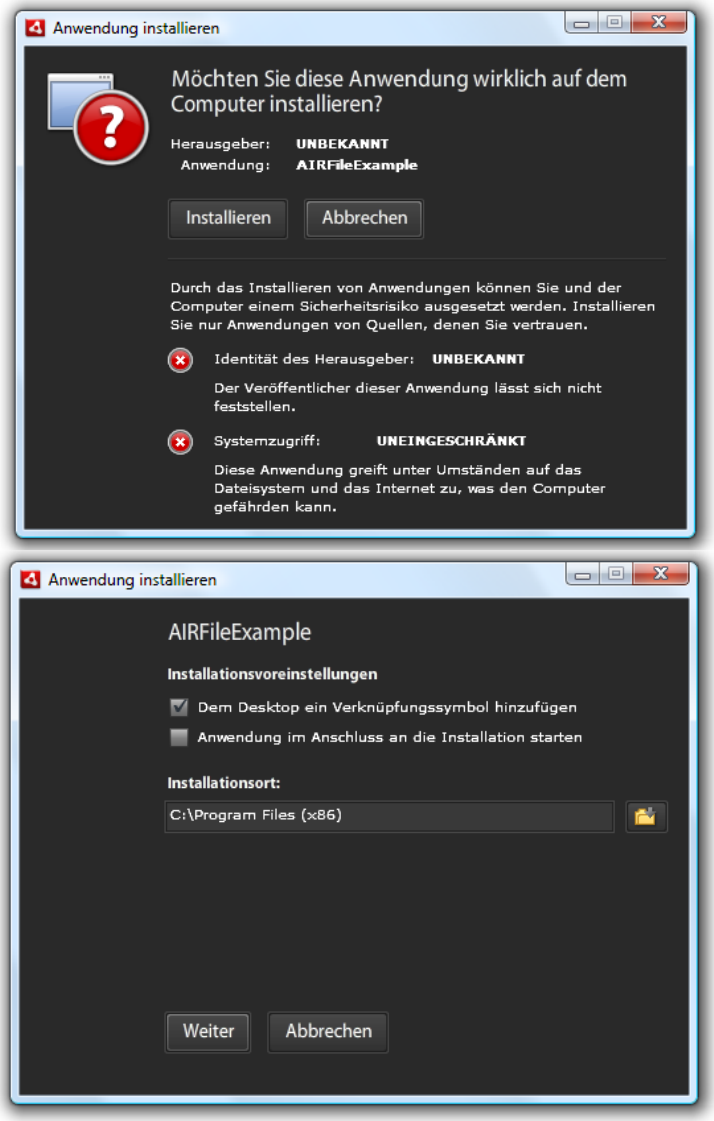


Figure 24: AIR Application Installation Process

The directory of the installed applications contains `.swf` sources, some manifest data and an `.exe` file. Double-click the `.air` installer or the `.exe` file to run the application.

5 Microsoft Silverlight

Silverlight is a cross-platform .NET runtime, cross-browser plug-in, and a set of Windows-based developer tools for building RIAs. At its heart, Silverlight is an implementation of the concepts and standards from Windows Presentation Foundation (WPF) such as binding, the property system, and Extensible Application Markup Language (XAML) in a cross platform version of the .NET Common Language Runtime (CLR) and libraries [Brown10].

5.1 Specification

Silverlight is compatible with today's sophisticated web browsers in combination with Microsoft Windows and Mac OS X. It is also the application development framework for Windows 7 Phone[35]. Although parts of earlier release versions were ported to Linux with the open source Moonlight[36] project, the SDK is still proprietary as of version 4 released on April 15 2010. The primary IDE for Silverlight applications (and generally applications targeted for the .NET Platform) is Microsoft's Visual Studio. With Visual Web Developer Express Microsoft offers a free, lightweight IDE using the Microsoft Essential Library for building Web applications with Visual Basic or C#[37].

Silverlight is basically a specialized subset of the .NET Framework and WPF for developing user-centric applications for the Web. WPF is used to build the Graphical User Interface of thick desktop and web application clients on the Windows platform. It was initially released as part of .NET Framework 3.0 released on November 06 2006. WPF separates the user interface from the business logic and employs the declarative XAML. Silverlight consists of two crucial parts: the Presentation Core and its own implementation of the .NET Framework[38]. While the Presentation Core includes UI rendering, layout, input handling and a subset of XAML - to name but a few - .NET for Silverlight includes data parsing and service communication functionality.

---

[35] http://www.silverlight.net/getstarted/devices/windows-phone/ – requested in January 2011
[36] http://www.go-mono.com/moonlight/ – requested in January 2011
[37] http://www.microsoft.com/express/Web/ – requested in January 2011
[38] cf. http://msdn.microsoft.com/en-us/library/bb404713%28VS.95%29.aspx – requested in January 2011

In contrast to WPF, Silverlight executes on a browser-hosted version of the Common Language Runtime (CLR).

5.1.1 Runtime Environment

The runtime environment of Silverlight is a derivative of Microsoft's CLR, an implementation of the Common Language Infrastructure (CLI). The CLI defines an infrastructure that is able to execute multiple high-level languages. The languages are compiled into the Common Intermediate Language (CIL). The infrastructure allows assemblies to run without modification on every platform the infrastructure is available on [LRWA08].

5.1.2 Language Characteristics

Similar to Adobe Flex Microsoft Silverlight utilizes multiple languages for different proposes. The Graphical User Interfaces is designed with a declarative language called XAML while one is encouraged to write the application's logic in any of the languages supported by Microsoft's .NET framework.

XAML

The eXtensible Application Markup Language (XAML) was introduced with WPF to contribute to the MVC pattern. XAML shares great similarity with Flex's MXML. The main purpose is to provide an easy to read and easy to write declarative language for GUI building tasks. Developing a GUI with markup requires significantly less code than with a standard object oriented programming language. Furthermore, the hierarchy of components can be easily visualized (and auto-coded) as it is derived from the nesting structure may applied to an application. Similar to MXML, XAML generates no abstract objects. In fact, each XAML element maps to a CLR object instance and each attribute maps to a CLR property.

XAML uses the root tag `<UserControl>` to define the content of a page – Silverlight uses a page-based metaphor – as a whole arbitrary control element as shown in listing 23.

```
<UserControl x:Class="SilverlightApplication7.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="Click Me!" Height="23" HorizontalAlignment="Left"
                Margin="60,48,0,0" Name="button1" VerticalAlignment="Top"
                Width="75" Click="button1_Click" />
    </Grid>
</UserControl>
```

Listing 23: XAML Example

Listing 23 also illustrates the `x scope` namespace. Namespaces in XAML basically follow the same principal as in MXML: they refer to packages. However this special namespace is used to join markup pieces of a class and to identify XAML elements for reference and use in code-behind. For example, in order to join any code-behind to an XAML file through a partial class (explained later), that class needs to be named as the `x:Class` attribute in the root element of the respective XAML file.

Code-Behind

Silverlight strictly separates view from logic in the development process, too. While the `.xaml` file describes the static GUI, all necessary interactivity programming has to be implemented in a corresponding `.xaml.cs` file. Also the initialization of the application takes place in the `.xaml.cs` file (cf. listing 24).

```
namespace SilverlightApp
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Clicked!");
        }
    }
}
```

Listing 24: Code-Behind Example

A Silverlight application may feature various partial classes splitting the definition of a class between different files (e.g. `.xaml` and `.xam.cs`). Such partial classes are assembled on compilation.

The great benefit of Silverlight is that is does not imply the use of a specific language for (logic) programming. Silverlight applications can be written in any .NET supported language like C#, C++, VB or JScript (a.k.a. JavaScript) as all code is compiled to CIL bytecode before being executed by a Virtual Machine. So, web developers may leverage their existing code expertise or veteran .NET developers can create Silverlight applications with their CLI language of choice without being forced to learn one specific language.

5.1.3 Deployment

The Silverlight application deployment package is a compressed file called the XAML Application Package (XAP). This file is simply a compressed ZIP archive that stores mandatory files, such as the application manifest and the main application DLL, and optional files, such as the auxiliary library DLLs and resource files. The application manifest file mainly includes a list of assembly files that need to be downloaded upon application startup [GhSc09].

Every Silverlight application additionally contains an `App.xaml` file defining resources like styles and brushes. The code-behind file of `App.xaml` holds startup instructions, e.g. which `.xaml` file to show initially. The Silverlight application can basically be deployed in two ways:

- Surfacing the `.xap` to the client via some URI by embedding it with the `<object>` tag in a HTML page
- Referencing the JavaScript utility file `Silverlight.js` and instantiating the Silverlight plug-in on the web page or within a hosting out-of-browser process:

Implementation via HTML object Tag

Listing 25 demonstrates how to embed a Silverlight application in a web page using the `<object>` element. The `type` attribute of the `<object>` tag identifies the plug-in and the required MIME type. The mandatory “source” `param` child element locates

the `.xap` deployment package. `onerror` specifies a JavaScript exception handler. The link is only displayed if Silverlight is not installed on the client machine.

```
<div id="silverlightControlHost">
  <object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2" width="100%" height="100%">
    <param name="source" value="MySilverlightRIA.xap"/>
    <param name="onerror" value="onSilverlightError" />
    <a href="http://go.microsoft.com/fwlink/?LinkID=141205"
        style="text-decoration: none;">
      <img src="http://go.microsoft.com/fwlink/?LinkId=108181"
          alt="Get Microsoft Silverlight" style="border-style: none"/>
    </a>
  </object>
</div>
```

Listing 25: Deploy Silverlight Application via object Tag

Implementation via Silverlight.js

The `Siverlight.js` utility file encapsulates methods and variables for embedding a Silverlight application into a web page. It is an integral part of every Silverlight application and is distributed along with the other application files via the `.xap` archive. In order to utilize the `Silverlight.js` utility file, reference it in HTML code of a web page at first (cf. listing 26).

```
<html>
  <head>
    <title> My Silverlight Application </title>
    <script type="text/javascript" src="Silverlight.js"> </script>
  </head>
  <body>
    <!-- Initialize Plug-in here. See listing 27.-->
  </body>
</html>
```

Listing 26: Reference Silverlight.js in HTML

Doing so gives access to the `createObjectEx()` function of the `Silverlight.js` utility file necessary for initialization of the Silverlight application. A `<div>` element like in listing 27 acts as the host of the plug-in instance. It is placed inside the `<body>` tag of an HTML page with a reference to `Silverlight.js` like the one from listing 26. The instantiation function `Silverlight.createObjectEx` takes three

mandatory parameters: `source`, `parentElement` and `id`. `source` specifies the `.xap` packages, `parentElement` the plug-in instance host and `id` provides a hook for HTML DOM access. The `properties` parameters clarify the initial appearance. If the Silverlight application is supposed to interact with HTML DOM (see section 5.4) it is crucial to set `enableHtmlAccess` to "true". `"events"` allows specifying `onload` and `onerror` event handler.

```
<div id="mySilverlightHost" style="height:100%;">
  <script type="text/javascript">
    Silverlight.createObjectEx({ #C
        source: "MySilverlightRIA.xap",
        parentElement: document.getElementById("mySilverlightHost"),
        id: "mySilverlightControl",
        properties: {
            width: "100%",
            height: "100%",
            version: "3.0",
            enableHtmlAccess: "true"
        },
        events: {}
    });
  </script>
</div>
```

Listing 27: Silverlight Plug-in Initialization

5.2 Data Binding

The ability to bind the state of a property to another is a vital part of the view-logic-separation paradigm. GUI and business logic can be developed independently. Data Binding in Silverlight is similar to that one in Flex. If Silverlight data sources (CLR objects) should propagate a change in state, however, it has to be enhanced with a change notification.

The basic data binding syntax is the following [CaGh10]:

```
<object targetPropertyname =
"{Binding sourcePropertyPath, oneOrMoreBindingProperties}" .../>
```

Every Silverlight binding statement starts with the so-called Binding markup extension and must specify a source and a target as shown in figure 25.

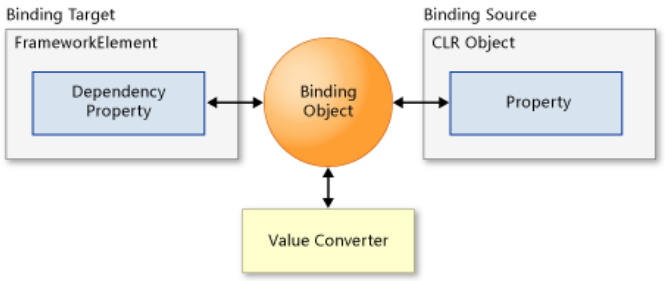


Figure 25: Silverlight Binding Scheme[39]

The target needs to be a `Dependency Property` – a special Silverlight extension to standard CLR properties. If the source is also a Dependency Property (XAML object property), just specify the property to bind and the source object as `ElementName`:

```
<TextBox Name="txtBindTarget" Text="{Binding Text, ElementName=txtBindSource}"/>
```

The optional value converter exposes the methods `Convert()` and `ConvertBack()` to modify data as its bound from the source object to the control and vice versa. CLR object properties of basic data type, though, first require the source object they belong to being set as `DataContext` property of the binding target before referencing it within the binding statement. The `DataContext` property allows binding entire collections to a visual control.

Silverlight supports three binding modes for regulating the data flow:

- OneTime: sets the target property only when the source is initialized
- OneWay: default binding mode; the target automatically receives updates from the source property but not vice versa
- TwoWay: mutual change propagation and updating

As already mentioned, CLR needs a change-notification handler for automatic update broadcasts. A change-notification handler notifies a binding target that a change has been made.

[39] http://msdn.microsoft.com/en-us/library/cc278072%28v=vs.95%29.aspx#Y798 – requested in January 2011

The `PropetyChanged` event of the `INotifyPropertyChanged` interface tells the binding engine that the source has changed so that the binding engine can update the target value. First declare the `PropertyChanged` event:

```
public event PropertyChangedEventHandler PropertyChanged;
```

Call a method `NotifyPropertyChanged()` as in listing 28 whenever the property is updated. If the `PropertyChanged` event is implemented, pass the name of the property subject to update as argument on `PropertyChanged()` invocation.

```
public void NotifyPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this,
                new PropertyChangedEventArgs(propertyName));
        }
    }
```

Listing 28: NotifyPropertyChanged method

Data Form

Examples in chapter 4 demonstrated that data grids are an essential control for structuring (bound) data. With the data form Silverlight provides a control focusing not only on the structured representation but especially on the manipulation of individual data. The following example is based on an application from [Brown10] and demonstrates how a collection of data – a student repository - can be efficiently connected to a UI for presentation to and manipulation by the user. A `Student` class like in listing 29 holds information about the students.

```
public enum AcademicDegree
    {
        None,
        BSc,
        MSc
    }
    public class Student
    {
        [Required]
        public int ID { get; set; }
        public AcademicDegree AcademicDegree { get; set; }
        [Required]
        public string LastName { get; set; }
        [Required]
        public string FirstName { get; set; }
        public bool ReRegistered { get; set; }
```

```
        public DateTime DateOfBirth { get; set; }
        public string EmailAddress { get; set; }

    }
```

Listing 29: Student Class

Note the `[Required]` attribute above some properties. Corresponding fields in the data form must not be left blank later on. A `StudentRepository` class creates an `ObserveableCollection` and fills it with items of `Student` type (cf. listing 30).

```
public class StudentRepository
    {
        private ObservableCollection<Student> _student =
        new ObservableCollection<Student>();
        public ObservableCollection<Student> Student
        {
            get { return _student; }
        }
        public StudentRepository()
        {
            _student.Add(new Student()
            {
                ID = 1234,
                AcademicDegree = AcademicDegree.None,
                FirstName = "John",
                LastName = "Dorian",
                ReRegistered = true,
                DateOfBirth = DateTime.Parse("1975-04-06")
            });
            _student.Add(new Student() {
                ID = 5678,
                AcademicDegree = AcademicDegree.BSc,
                FirstName = "Elliot",
                LastName = "Reid",
                ReRegistered = true,
                DateOfBirth = DateTime.Parse("1976-08-27")
            });
        }
    }
```

Listing 30: StudentRepository Class

The next step is to create the `DataForm` and reference and integrate the `Student` and the `StudentRepository` classes.

This requires adding the `StudentRepository.cs` depicted in listing 30 as static resource identified via “repository” to the `MainPage.xaml` of the application:

```
<UserControl.Resources>
        <local:StudentRepository x:Key="repository" />
</UserControl.Resources>
```

The newly declared static resource will be assigned to the `DataContext` property of the `DataForm` XAML control while the `ItemsSource` of the `DataForm` will be the collection with objects of class “Student” exposed through a property named “Student” from the `StudentRepository` (cf. listing 31).

```
<toolkit:DataForm Name="dataForm1"
           DataContext="{StaticResource repository}"
           ItemsSource="{Binding Student}"
           CurrentIndex="0"
           CommitButtonContent="Save"
           CancelButtonContent="Cancel"/>
```

Listing 31: Silverlight DataForm control

The `DataContext` refers to a top-hierarchy object type. It allows binding multiple containing properties to the UI. `ItemsSource` defines objects from the `DataContext` which can be bound to values of the control. The data form automatically assigns each property of the `ItemsSource` to a row in the form and renders it accordingly to its data type as shown in figure 26. For instance, the `DateOfBirth` property is rendered with a trailing date picker.

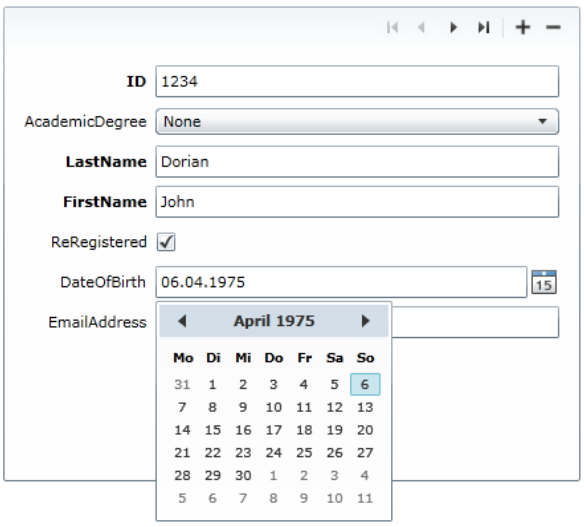


Figure 26: Silverlight Data Form Entry

What is more, the data form features some built-in data validating. Alerts are shown in case of syntactically incorrect input or blank fields marked as required (cf. figure 27).

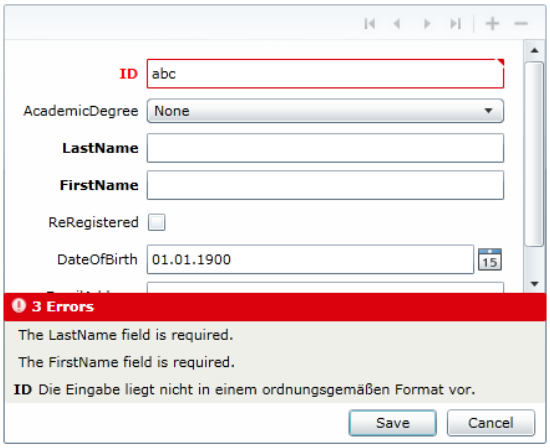


Figure 27: Data Form: Built-In Validating

Unfortunately, Silverlight – to be more precise XAML – lacks specialized validator elements like the `<mx:EmailValidator>` from the Flex SDK. You are supposed to write validators on your own. Nevertheless, the `System.ComponentModel.DataAnnotations` package can be leveraged to set up common validation rules, e.g. to check input against defined regular expressions.

5.3 Remote Service Handling

In Silverlight external resources on the Web can be basically accessed with two mechanisms:

- `WebClient` class:

  The `WebClient` type has a convenient collection of methods that let you access resources over HTTP. You can use the `WebClient` class in two basic modes: uploading/downloading resources as strings and reading from or writing to streams [CaGh10].

- `HttpWebRequest` class:

  The `HttpWebRequest` is designed to communicate over the HTTP and HTTPS protocols. It also supports the POST method along with GET, whereas `WebClient` only supports GET [GhSc09].

Nevertheless, you are always at liberty to leverage the `XMLHttpRequest` object of AJAX trough interaction with JavaScript instead.

To call the service using `WebClient`, like in listing 32, start with the declaration of a `WebClient` object. Then associate the `OpenReadCompleted` event of the `WebClient` with a method for callback. Finally initiate the request with the desired URI.

```
WebClient client = new WebClient();

/* The event assignment "+=" is a C# feature. It sets (adds) an event hand-
ler for the OpenReadCompleted event. The opposite is "-=" which removes a
certain event handler. */
client.OpenReadCompleted +=
          new OpenReadCompletedEventHandler(client_OpenReadCompleted);

client.OpenReadAsync(new Uri(http://localhost/sampleURI));

void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
    // response parsing
}
```

Listing 32: Using WebClient for Resource Request

Listing 33 shows the different approach, using `HttpWebRequest`. An instance of `HttpWebRequest` cannot be created directly. The `WebRequest` class contains a factory method named `Create()` that returns an appropriate instance of a `WebRequest` inheritor, based on the protocol specified in the URI [GhSc09].

Invoke `BeginGetResponse()` on the request, pass the callback method as a new `AsyncCallback` object with the request object as the state parameter. In the callback method, get access to the request object via the `AsyncState` parameter and get the response object from `EndGetResponse()`, which ends the request. In order to get access to the body of the response, retrieve the response stream and read it with a `StreamReader`.

```
HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(new
Uri(http://localhost/sampleURI));

request.BeginGetResponse(new AsyncCallback(ReadCallback), request);

private void ReadCallback(IAsyncResult result)
{
    HttpWebRequest request = (HttpWebRequest)result.AsyncState;

    HttpWebResponse response =
          (HttpWebResponse)request.EndGetResponse(result);

    StreamReader reader = new StreamReader(response.GetResponseStream()))
}
```

Listing 33: Using HttpWebRequest for Resource Request

A possible service handling workflow is outlined in figure 28. An instantiated Silverlight application uses `WebClient` to prepare and send a request to a server (e.g. Java EE server). On the server the appropriate service is invoked and returns a response. Now it is the client application's turn to parse the response for further processing.

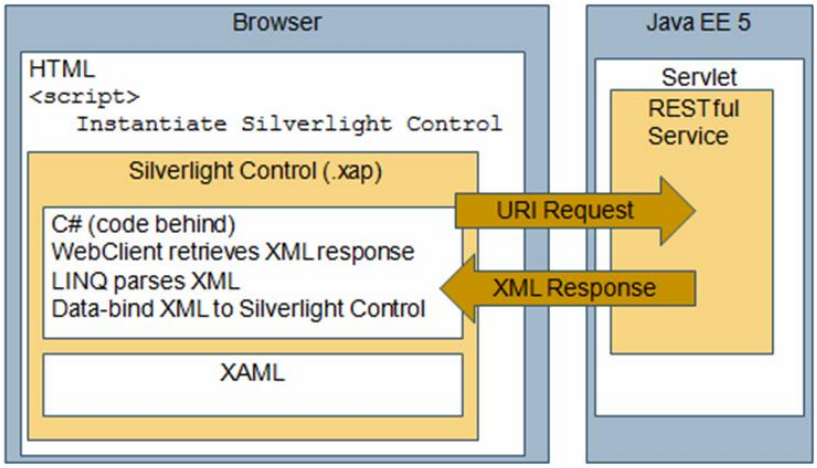


Figure 28: Silverlight Service Consumption Example Workflow[40]

If the response data is already strongly typed – using a SOAP service for instance – it can be integrated directly. In most cases the response contains raw XML.

[40] http://www.infoq.com/articles/silverlight-java-interop – requested in January 2011

Three different built-in methods can be used in Silverlight to parse XML:

- LINQ to XML (Language Integrated Query to XML – formerly called XLINQ)

  LINQ to XML is a dialect of the .NET Framework's LINQ. LINQ to XML has data querying capabilities and uses query syntax to access the nodes and attributes in XML. It makes working with data such as collections of objects and XML documents much easier than with DOM[41]. The first step is to create an `XmlReader` from the response stream:

  ```
  XmlReader responseReader = XmlReader.Create((Stream)stream);
  ```

  That `XmlReader` is loaded into a `XElement`:

  ```
  XElement xmlResponse = XElement.Load(responseReader);
  ```

  Any element or attribute can subsequently be accessed by name to get its value:

  ```
  XElement item = xmlResponse.Element("item");
  ```

- XmlReader

  The `XmlReader` class is a fast-forward–only, non-caching XML parser. For processing large XML files, `XmlReader` is better suited than LINQ to XML for performance reasons [Brown10]. The following code from listing 34 leads to the same result as with LINQ to XML:

  ```
  XmlReader responseReader = XmlReader.Create((Stream)stream);
  responseReader.Read();
  responseReader.ReadToFollowing("item");
  String item = responseReader.ReadElementContentAsString();
  ```

  Listing 34: Parse Response Data with XMLReader

- XmlSerializer

  The `System.Xml.Serialization` namespace provides the `XmlSerializer` class that one can use to easily save and load objects to any stream. The `XmlSerializer` provides a way to convert an `XmlReader` into strongly typed

[41] See http://msdn.microsoft.com/en-us/library/bb387021.aspx – requested in March 2011 – for a comparison between LINQ to XML and DOM.

objects. To use this approach, one needs to define a class that matches the format of the incoming XML. The workflow is the same as with `XmlReader` until it points to the requested element. Next create a class equivalent to the element's structure. Then create an `XMLSerializer` with the target type:

```
XmlSerializer serializer = new XmlSerializer(typeof(item));
```

Deserialize the `XmlReader` to get an object of the target type:

```
item i = (item)serializer.Deserialize(responseReader);
```

Now get access to child elements of the serialized parent element as they were properties of it. For instance, the (de)seralized object `i` has an element `name` which should be assigned to a variable `name`. This is accomplished via:

```
String name = i.name;
```

Although Silverlight is featuring the discussed powerful ways to connect to external services and to parse response data, Visual Studio does not excel that much as Flex does with Flash Builder when it comes to wizard-supported, automatic service detecting and integration. Basically you can automatically integrate a service via a so-called Service Reference. Right click on a project and select "Add Service Reference".

The dialog, shown in figure 29, allows you to enter the URI of a service. Click "Go" in case of an external service or "Discover" in case of a service part of the same solution to analyze the signature of the service. If the URI points to a valid service, all available operations are listed and can be implemented on the basis of a proxy mediating communication between you and the remote service.

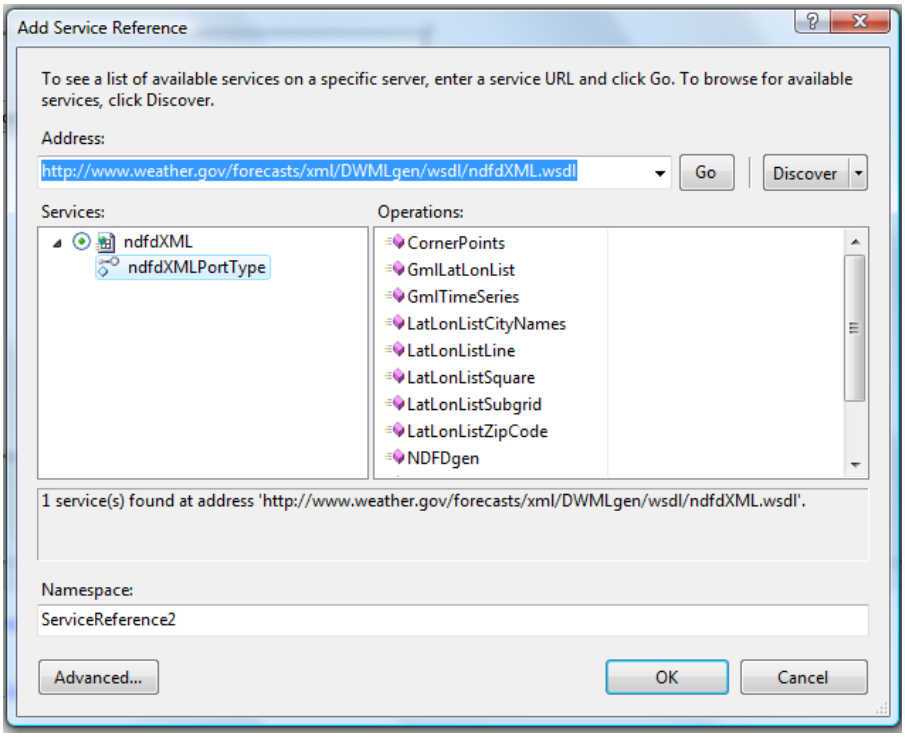


Figure 29: Reference Service in Silverlight

This sounds pretty much like the data wizard functionality in Flash Builder. Unfortunately, the auto-generating part ends with the creation of the proxy. In contrast to Flex in combination with the Flash Builder, in Visual Studio one has to invoke methods on the proxy manually, has to write event handlers with callbacks and has to take care of integrating the result data with the UI.

5.4 Interaction with the Environment

While Flex is on the leading edge concerning efficient service integration, Silverlight performs outstandingly when it comes to interaction with the HTML environment. As discussed in section 4.4, the spectrum of communication from Flex to JavaScript and vice versa is limited to explicitly “exposed” functions and special variables. In contrast to an embedded Flex/Flash application, an embedded Silverlight application is able to directly interact with the DOM, with JavaScript functions and even with the browser object itself. So the HTML environment can be directly addressed from inside managed code (Silverlight). The prerequisite is to set the `enableHtmlAccess` property to “true” when instantiating the plug-in on a web page (cf. section 5.1).

The following tasks can be accomplished with this functionality known as HTML Bridge[42]:

- Manage the web page from managed code:

  Use the root `HtmlElement` of the `document` property of an `HtmlPage`, a reference to the body of the HTML document, and methods for retrieving elements on the page by ID. This enables you to navigate the contents of a web page, work with individual element properties and styles, and retrieve information from a query string [Brown10]. For example get access to a certain element via its ID:

  ```
  HtmlElement element = HtmlPage.Document.GetElementById("myDiv");
  ```

- Work with the browser window:

  The `HtmlWindow` class provides a direct connection to the functionality of the browser, including shortcuts to display alert and confirmation dialogs, use navigation controls, execute arbitrary script code, and access bookmarks within the page [GhSc09]. Address it through the `Window` property of the `HtmlPage` object and invoke certain functions on it:

  ```
  HtmlPage.Window.Alert("Hello World!");
  ```

- Calling Silverlight from JavaScript:

  The first step is to mark a class with the `ScriptableType` attribute. This attribute, which is part of the `System.Windows.Browser` namespace, makes a class accessible to JavaScript:

  ```
  [ScriptableType]
  public partial class MainPage : UserControl
  ```

  Once a class has been marked as a `ScriptableType`, all public properties, methods, and events are available to JavaScript.

[42] The cross-browser compatibility of the HTML Bridge is derived from the cross-browser compatibility of its functional assets (DOM, JavaScript). Hence, the HTML Bridge basically works fine in combination with all modern browsers. See http://www.quirksmode.org/dom/w3c_core.html – requested in April 2011 – for a brief compatibility check.

Use the `ScriptableMember` attribute to expose selected events, methods or properties to JavaScript:

```
[ScriptableMember]
public void bridgeMethod()
{}
```

After the scriptable objects are defined, they have to be registered for use in JavaScript by invoking the `RegisterScriptableObject()` method on the `HtmlPage` class:

```
HtmlPage.RegisterScriptableObject("objectAlias", objectInstance);
```

The object is registered with the scripting engine by passing it as the second parameter to the `RegisterScriptableObject()` method, which then uses the first parameter to create an alias for the class instance [Brown10]. This alias is appended to the `content` property of the hosting Silverlight plug-in:

```
var silverlight = document.getElementById("mySilverlightRIA");
silverlight.content.objectAlias.bridgeMethod();
```

- Calling JavaScript from Silverlight:

  Silverlight gives you the flexibility to call JavaScript from managed code and, in turn, the ability to call any method on an HTML or JavaScript object via the `Invoke()` method [Brown10]. The `Invoke()` method can be applied to any `HtmlDocument`, `HtmlElement`, or `HtmlWindow` object. The first parameter of this method represents the name of the function to be invoked. The trailing parameter(s) represent(s) the argument(s) that will be passed to this function:

```
HtmlPage.Window.Invoke("jsFunction", "argument");
```

Now image a scenario in which it is necessary to incorporate both Flash and Silverlight technology in a web application. The following nutshell puts their interoperability to the test.

Calling Flash from Silverlight

The very first step is to set up the callback in the Flex project:

```
ExternalInterface.addCallback("sayHello", sayHello);
```

Invoke the JavaScript function which propagates the function call to the Flash object:

```
HtmlPage.Window.Invoke("callFlash");
```

Finally, the invoked JavaScript function gets the already in HTML embedded Flex application and calls the appropriate Flex function:

```
function callFlash ()
{
    flashApp = document.getElementById("flashApp");
    flashApp.sayHello();
}
```

Call Silverlight from Flash

Mark a type or member as scriptable

```
[ScriptableType]
public partial class bridgePage : UserControl
{}
```

and register it to the script engine:

```
HtmlPage.RegisterScriptableObject("bridgePage", this);
```

Invoke the JavaScript function which propagates the function call to Silverlight:

```
ExternalInterface.call("callSilverlight");
```

Finally, the invoked JavaScript function calls the appropriate Silverlight function on the embedded and instantiated Silverlight object:

```
function callSilverlight()
{
    silverlight.content.bridgePage.sayHello();
}
```

The bottom line is that JavaScript is able to link Flash and Silverlight. They are not mutually exclusive but offer synergy potentials to a certain extent. For example, a DHTML-based legacy front-end can be gradually extended with Flash-based multimedia content and Silverlight-based forms integrating WCF services from a .NET back-end.

5.5 Silverlight Out-of-Browser

Silverlight 3, released on July 9 2009, introduced the feature to install Silverlight applications on the local client machine. This Out-Of-Browser (OOB) activation model neither requires an active connection to the web source of the application nor an open browser window.

Out-of-browser Silverlight applications work just like in-browser Silverlight applications with some minor differences [Brown10]:

- Isolated storage quota for out-of-browser applications is 25 MB by default as opposed to 1 MB for in-browser applications. In both cases, this can be extended by prompting the user.
- Out-of-browser applications provide access to keys that the browser normally captures, such as function keys.
- Out-of-browser applications can be pinned to the Start menu or taskbar on Windows systems and display custom icons.
- Out-of-browser applications require an explicit check for a new version, whereas in-browser versions automatically update.
- Out-of-browser applications support the elevated trust mode (see below).
- Use Component Object Model (COM) automation to integrate with native code and applications on the desktop.
- Out-of-browser applications can’t receive initialization parameters or take advantage of any of the plug-in parameters while running out of the browser.
- Out-of-browser applications can’t interact with the HTML DOM - there’s no DOM to work with.

With version 4 of Silverlight released on April 15 2010 it is possible to substitute the Windows chrome[43] with a custom one. For every Silverlight application one has the choice to substitute the default chrome style with a no border or a borderless round corner style. In addition, Silverlight features events and functions which support building custom chrome.

At a glance, OOB seems to be very similar to AIR, but they are very different actually. OOB applications do not run in a separate runtime environment because every Silverlight application can easily configured as OOB (camouflaged as desktop application) and OOB does not require extended APIs for working with local resources as it is basically neither intended nor privileged to work with them. What is more, OOB applications are still bound to the sandbox[44]. This means that they are subject to strict limitations on what system resources and services they can access or request in order to prevent accidently or maliciously intended harm to the client machine.

Silverlight 4 introduced an elevated trust system, though. If the deployment package (`.xap`) is signed with a certificate of a trusted authority, the application can escape the sandbox to a certain extent. When trusted, the application is privileged to read and write files in the user document folders and no longer has to use proxies for cross-domain web service calling, for example. However, trusted OOB applications still do not enjoy the rigorous access to the local storage system as AIR applications do.

Let's have a look at the installation process of OOB. Earlier versions of Silverlight required editing the `AppManifest.xml` file in order to setup your application for OOB. With the release version 4 just go to the "Properties" section of the project and tick "Enable running application out of the browser" as shown in figure 30.

Then click on the now accessible "Out-of-Browser Settings …" button to configure the displayed title, icon and the like. If you tick the option for elevated trust, your application will not be installable until it is signed.

---

[43] The chrome of a GUI refers to the borders, title bars, buttons (minimize, maximize and close) and other elements that decorate a typical window on a given operating system.

[44] See http://searchsecurity.techtarget.com/definition/sandbox – requested in April 2011 – for more information on basic sandbox concepts.

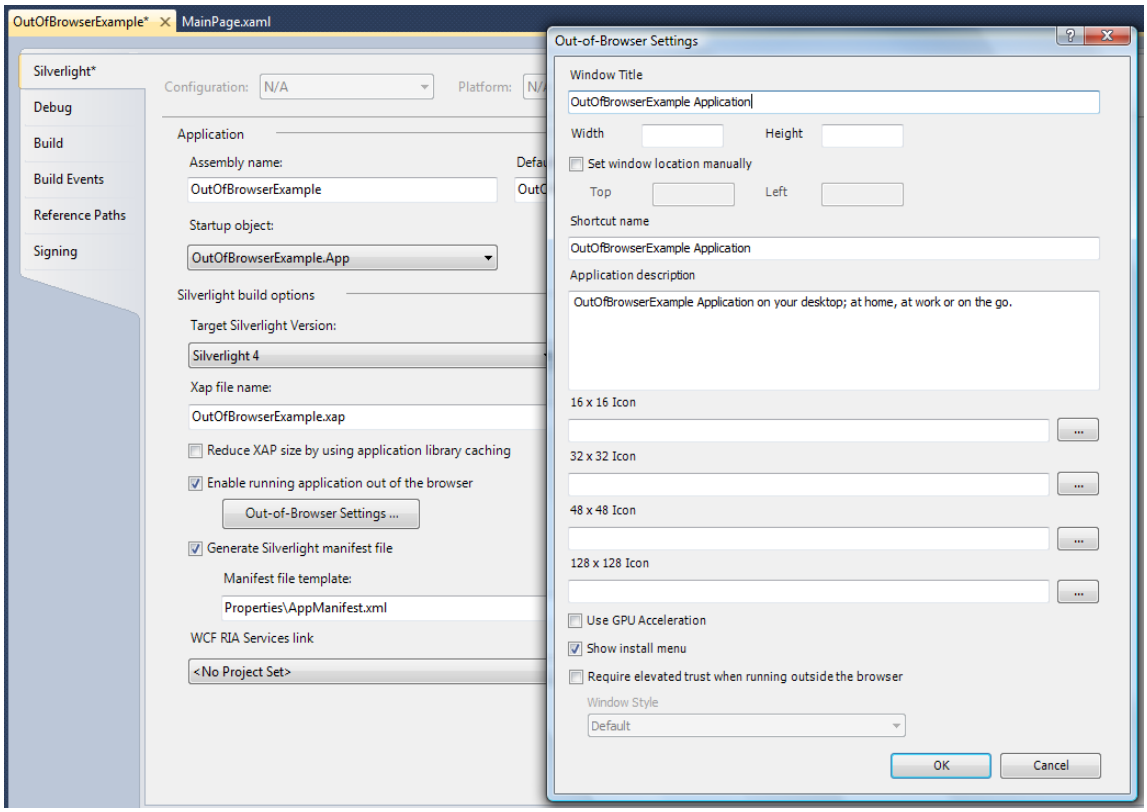


Figure 30: Silverlight OOB Setup

As soon as you confirm the OOB settings, an `OutOfBrowserSettings.xml` file holding the configuration is added to the properties of your application. The installation is initiated with a right click on the application running in the browser and selecting the install command from the context menu. Another option is to trigger the `Application.Current.Install()` method via UI event, e.g. the user initiates the installation by clicking on a button of the GUI wired up with a respective event handler (cf. figure 31).

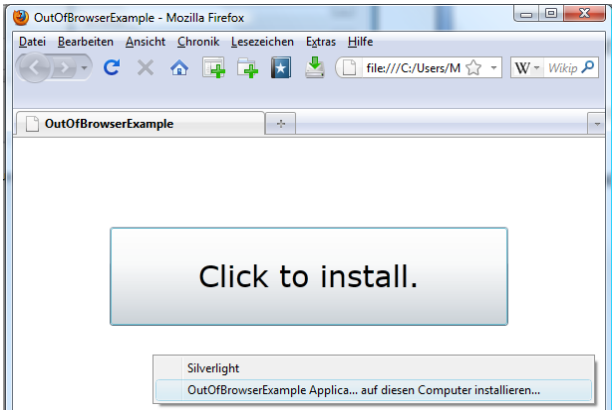


Figure 31: OOB installation

The property `Application.Current.IsRunningOutOfBrowser` allows observing the state of the application. For instance, provide different functionality or show different interfaces. Figure 32 illustrates both the displayed interface of the application from figure 31 after it is installed and the interface of the desktop version.

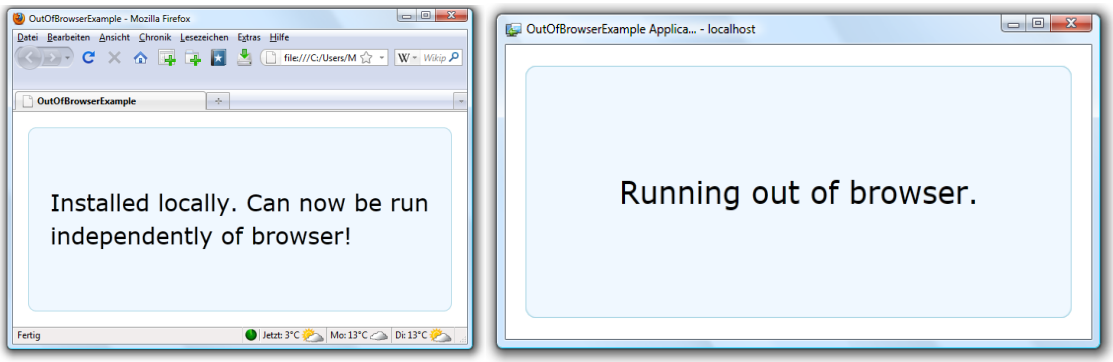


Figure 32: State dependent User Interfaces

6 JavaFX

JavaFX, the youngest of the discussed RIA frameworks, is developed by Sun Microsystems. The technology formerly known as F3 (Form Follows Function) is essentially the successor of Java Applet. While Java is indisputably omnipresent in the back end (server tier), it is not the first choice for the frontier view of modern web applications. Writing a GUI in an XML-based language just requires significantly less code and time than with Java's Abstract Windows Toolkit (AWT) or Swing, for example. In order to overcome this potential weak spot, JavaFX is designed to tremendously simplify the creation of graphical user interfaces on the Java platform.

6.1 Specification

Using the Java platform at its core, JavaFX works seamlessly with the Java platform and can easily leverage existing Java code. This also allows JavaFX to leverage the "write once, run anywhere" capability provided with the Java platform [WGCI09]. The JavaFX 1.3 SDK, released on April 22 2010, can be downloaded from `http://javafx.com`. Plug-ins for both NetBeans and Eclipse IDE exist and enable syntax highlighting, code completion, debugging, etc in JavaFX projects. Adobe Photoshop and Adobe Illustrator offer integration via JavaFX Production Suite plug-ins. These plug-ins enable you to generate JavaFX data files from created graphics.

6.1.1 Runtime Environment

JavaFX runs on any system and browser as long as a Java Runtime Environment (JRE) is installed. The version targeted at mobile devices is called JavaFX Mobile and runs on top of the Java Micro Edition (ME). Unfortunately JavaFX Mobile is trailing comparable technologies like the iPhone SDK or Android SDK[45].

[45] http://mobile360.de/javafx-mobile-hebt-nicht-ab-32351.html (German) – requested in February 2011

6.1.2 Language Characteristics

In contrast to Flex or Silverlight, JavaFX employs only one language: JavaFX Script. JavaFX Script combines full object orientation with a declarative syntax which is not based on XML but rather looks similar to CSS.

JavaFX Script

At a glance, the characteristics of JavaFX Script are [WGCI09]:

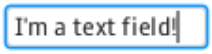
- It is an object-oriented language. It supports classes, instance variables, instance functions and - in contrast to Java – multiple inheritance via mixin classes.
- It supports a declarative style suitable for GUI programming. Its object literal syntax and sequence syntax make describing GUIs easy.
- It supports data binding. It allows for easy separation of GUI views and models.
- It is a statically typed, compiled language with basic type inference capabilities. It compiles source code into Java classes.
- It employs libraries for timelines and path-based animation similar to Adobe Flash.
- It can leverage the vast number of Java libraries. JavaFX classes can extend Java classes and interfaces, and JavaFX code can instantiate Java objects and call Java methods.

JavaFX Script is broken down into two main levels, script and class. The difference between script and class level is that variables and functions defined at the script level are globally accessible inside the `.fx` project whereas class level follows the same scope concept as in Java. One exception to Java is that JavaFX employs no “static” prefix; script level fields and methods serve basically the same purpose. One may instantiates a class to get access to its properties.

In JavaFX objects are instantiated using object literals. This is a declarative syntax using the name of the class that you want to create followed by a list of initializers and definitions for this specific instance enclosed in curly braces “{}”.

For instance, listing 35 shows the creation of an instance of the class `javafx.scene.control.TextBox` via object literal syntax. Every instance variable initializer is followed by a colon “:” and a value or binding expression. Object literals may also consist of variable declarations and other object literals assigned to an instance variable. A close look on listing 35 actually reveals two object literals: the obvious outer one (“TextBox”), an instance of the `javafx.scene.control.TextBox` class, has another object literal (“Font”) of class `javafx.scene.text.Font` assigned to its `font` property.

```
TextBox {
      var mytext = "I'm a text box! "
      text: mytext
      columns: 12
      selectOnFocus: true
      font: Font {
          size: 14
      }
      layoutX: 100
      layoutY: 20
}
```



Listing 35: JavaFX Instance Creation via Object Literal

Instances of objects can be also created with the `new` expression, but for the sake of consistency you should still use curly braces “{}” instead of round braces. Listing 36 demonstrates the conventional approach using `new` for instantiation that looks more like Java. Please note the line with the command supposed to set the size of the `myfont` object. The `size` property of the class `javafx.scene.text.Font` has the `public-init` access modifier which allows no post initialization value assignment.

As JavaFX features no Java-like constructors per se – when using the `new` expression the class name is always followed by empty braces – trying to set the size of an instance of type `javafx.scene.text.Font` will result in a compile-time error, unless one uses object literals.

```
var textBox = new TextBox();
var mytext = "I'm a text box!";
textBox.text = mytext;
textBox.columns = 12;
textBox.selectOnFocus = true;
var myfont = new Font();
myfont.size = 14; // Compile-Time Error!!!
textBox.font = myfont;
textBox.layoutX = 100;
textBox.layoutY = 20;
```

Listing 36: JavaFX Instance Creation via Java Convention

Nevertheless, creating instances of Java object in JavaFX still requires the standard convention. Note that the `mytext` variable from listing 35/listing 36 obviously is of `String` data type and is recognized as such from the context.

Although you can declare a type, it is basically inferred from the first assigned value. Table 5 demonstrates additional vital features of JavaFX Script either not present in Java or at last differently operationalized.

Feature	Code Sample	Description

Sequences	/* initialize sequence of specific size (six) with items */
	`def footballPos = ["QB", "TE", "WR", "FS", "SS", "CB"]`
	/* initialize empty sequence of specific size (six) */
	`def footballPos = [0..12 step 2]`
	// insert "HB" after "QB"
	`insert "HB" after footballPos[0]`
		Ordered lists of objects, similar to arrays. JavaFX uses meaningful, reserved keywords like "insert" and "delete" list manipulation and "step" to set up intervals. Data type of sequence is inferred from "logical" data type of items.

Block expressions	/* declare a variable x and initialize it with the value 7 */
	`var x = {`
	`  var a = 5;`
	`  var b = 2;`
	`  a + b };`
		The block expression is formed by enclosing other expressions within a pair of curly braces.
		The type of the block expression is the type of the last expression it encloses. If the last expression is not of the void type, then the value of the last expression is the value of the block expression [WGCI09].

Condition expressions	// short form if-expression
	`var x = if (i == y) then 2 else 4;`
	
	// complex if-expression
	`var salary = if(rating < 80.0) 2.50 else if(rating < 120.0) 3.50 else 5.00;`
		Special type of block expression. Introduced by the `if` keyword, followed by a parentheses enclosed condition, which must be an expression of type `Boolean`; an optional `then` clause after the closing parenthesis; and an optional `else` clause.

For loops	// iterate over sequence
	`for(item in sequence) { println(item); }`
	
	// iterate over a count
	`for (i in [0..100]) { }`
		Introduced by the `for` keyword, followed by a pair of parentheses that enclose one or more comma-separated `in` clauses, and an expression after the closing parenthesis. `For` expressions are often used to loop over sequences or to create new sequences by returning an object on each iteration.

Functions and implicit return type	/*anonymous function with implicit return type */
	`var passRating: function(td:Number, att:Number);`
	`passRating = function(td, att) { td/att *20 };`
	
	// implicit return type
	`public function passRating(td:Number, att:Number) {`
	//return type of Number inferred
	`td/att*20 }`
		In JavaFX, functions are objects of themselves and may be assigned to variables. For example, to declare a function variable, assign a function to that variable, and then invoke the function through the variable.
		
		If return type of function is omitted, it is inferred from the last expression in the function expression block. Cannot be applied on recursive functions and two or more functions referencing each other.

Table 5: Selected JavaFX Script Features

With the release of JavaFX 2.0 (scheduled for late 2011) JavaFX Script will be discontinued as JavaFX moves to the regular Java API. However, functionality like data binding and the use of object literals is supposed to persist and should be exposed as library. The goal is to ensure a sophisticated integration with other Java UI toolkits and to encourage developers creating JavaFX applications with their

JVM language of choice. Since JavaFX Script is a very potent language, it presumably will be enhanced in open source projects of JavaFX[46].

6.1.3 Deployment

JavaFX applications can be deployed on the Web as a Java applet or via Java Web Start[47]. Java applets run inside the browser, are loaded on web page initialization and require a full download of necessary files on each startup. Java Web Start applications can run outside the browser, are only invoked on user interaction and cache necessary files on the client machine. Java applets were always criticized for long loading phases. Since the release of Java SE 6 Update 10 in October 2008, however, the applet framework is overhauled and leverages the local caching principle of Java Web Start. New data is only downloaded when the version of the `.jar` file is subject to update. The basic steps to deploy a JavaFX application as an applet or with Java Web Start are[48]:

1. Create and sign the JAR application file.

2. Create the JNLP (Java Network Launch Protocol) descriptor file.

3. Add JavaScript code to the HTML source or – Java Web Start exclusively - associate the Java Web Start MIME type and double click the `.jnlp` file[49].

Figure 33 shows the obligatory parameters which have to be set for deployment. In NetBeans go to the “Properties” of your project, select “Application” and tick “Self Signed Jar” and “Pack200 Compression”. Then switch to “Run” and mark “Run in

---

[46] An open source JavaFX Script utility project called JFXtras can be found at http://code.google.com/p/jfxtras/ – requested in April 2011. Oracle announced that all JavaFX UI controls will be released to open source with the release of JavaFX 2.0.
See http://drdobbs.com/blogs/java/228701550 – requested in April 2011 – for more information.
Additionally, Stephen Chin, co-author of [WGCI09] requests with a petition http://steveonjava.com/javafx-petition/ – requested in April 2011 – open sourcing the whole JavaFX platform.
Project Visage http://code.google.com/p/visage/ - requested in May 2011 – shall adopt the legacy of JavaFX Script outside of Oracle’s control.

[47] See http://www.java.com/en/download/faq/java_webstart.xml and http://www.oracle.com/technetwork/java/javase/overview-137531.html – both requested in March 2011 – for more information on Java Web Start.

[48] cf. http://download.oracle.com/javase/tutorial/deployment/applet/deployingApplet.html – requested in March 2011

[49] The MIME type of Java Web Start is normally auto-associated when Java is installed.

Browser" for applet deployment or "Web Start Execution" when using Java Web Start.

After the project is built, a `.jar` file, containing the application and a `.jnlp` file are created. Java Network Launching Protocol (JNLP)[50] was first introduced with Java Web Start. This feature defines the required resources, security settings, and applet configuration properties for deployment. It is now being applied to applets themselves, so that applets no longer run within the browser application, but rather run in a background Java process with a window on the browser page. [CCB09].

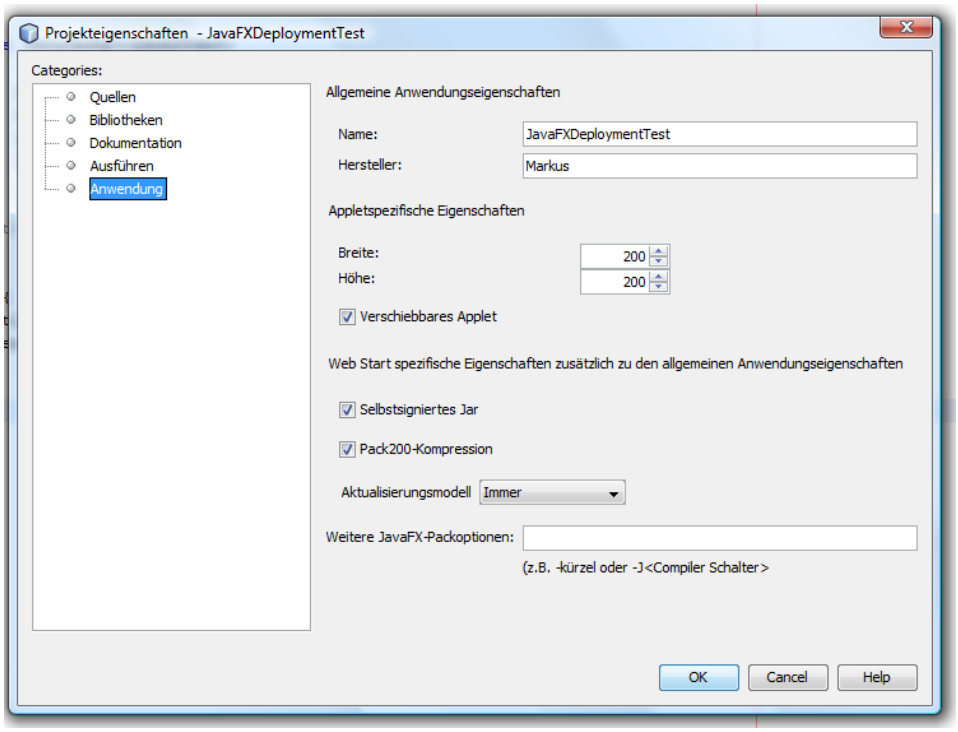


Figure 33: Applet/Web Start Generation Parameters in NetBeans

Finally the applet needs to be launched via a JavaScript function. If one is using NetBeans or any respective IDE this very last task will be done by the IDE as it auto-generates the needed code. However, if one is not using an IDE, the required dtfx.`js` file has to be referenced manually and its `javafx()` function also needs to be manually invoked with the initialization parameters (cf. listing 37 with figure 33).

[50] See http://java.sun.com/javase/technologies/desktop/javawebstart/download-spec.html - requested in March 2011 – for the JNLP specification.

```
<script src="http://d.javafx.com/1.3/dtfx.js"> </script>
<script>
javafx(
    {
      archive: "JavaFXDeploymentTest.jar",
      draggable: true,
      width: 200,
      height: 200,
      code: "javafxdeploymenttest.Main",
      name: "JavaFXDeploymentTest"
    }
);
</script>
```

Listing 37: JavaScript Code for Java Applet Deployment

Note that the `draggable` parameter has to be set to “true” if the application should be fit for browser undocking.

With Java Web Start the application is even simpler to reference in a web page. Just create a link pointing to the `.jnlp` file:

```
<a href="dist/JavaFXDeploymentTest.jnlp">Launch with Java Web Start</a>
```

Undock From Browser

With Java SE Version 6 Update 10, released on October 15 2008, it is possible to undock an applet from the browser. Provided that the aforementioned `draggable` property is set to “true” on deployment, you can click on an applet and drag it out of the browser onto the desktop while holding down the `ALT` key similar to the demonstration in figure 34. The applet persists, even if the browser is closed. Clicking on a close button (right corner “X” by default) integrates the applet in the browser once again.

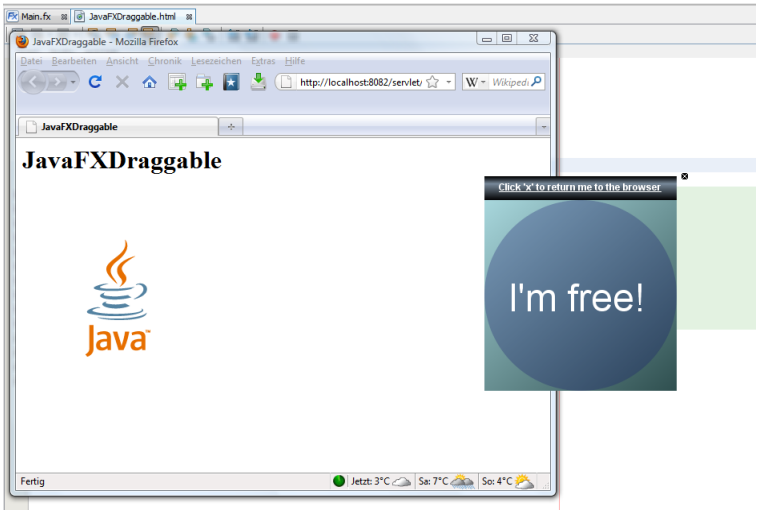


Figure 34: JavaFX Draggable Applet Example[51]

Add an instance from `javafx.stage.AppletStageExtension` to the extensions instance variable of your application GUI as demonstrated in listing 38 to define further behavior listed in table 6.

```
Stage {
        title: "A Draggable Applet"
        width: 450
        height: 330
         …
        extensions: [
            AppletStageExtension {
                //Defines mouse state if cursor is in draggable area
                shouldDragStart: function(e): Boolean {
                    return e.primaryButtonDown and dragArea.hover;
                }
                //Indicates when applet is out of the browser
                onDragFinished: function(): Void {
                    inbrowser = false;
                }
                //Indicates when applet is in the browser
                onAppletRestored: function(): Void {
                    inbrowser = true;
                }
                //Display default close button and use default close logic
                useDefaultClose: true
            }
        ]
    }
```

Listing 38: Using `javafx.stage.AppletStageExtension` to Define Dragging Behavior

[51] cf. http://download.oracle.com/javafx/1.3/tutorials/ShouldDrag/index.html – requested in February 2011

Extension Property	Description
appletDragSupported	Set to “true” if browser and Java installation support applet dragging.
onAppletRestored	Function, called when close button is pressed and applet returned to the browser.
onDragFinished	Function, called when applet is dragged to the desktop and the primary mouse button is released.
onDragStarted	Function, called when primary mouse button is held down and drag process starts.
shouldDragStart	Function, called when primary mouse button is held down on applet.
useDefaultClose	Determine whether default close button shall be displayed and default close behavior or own closing display object and logic should be used.

Table 6: Properties of `javafx.stage.AppletStageExtension`[52]

6.2 Data Binding and Triggering

As already discussed, data binding is a convenient way to connect the model of an application to the presentation view. JavaFX also incorporates the ability to synchronize the state/value of two objects or variables. However, data binding in JavaFX is very fine-grained and complex.

But data binding is not the only way to encapsulate change notifiers and getter and setter methodology. With triggers it is possible to attach a block of code to a variable executed each time the variable changes.

52 http://download.oracle.com/docs/cd/E17802_01/javafx/javafx/1.3/docs/api/javafx.stage/javafx.stage.AppletStageExtension.html – requested in February 2011

Table 7 discusses the nine (!) different ways to observe and automatically update dependent variables in JavaFX; from simple variable binding to binding of whole function closures (cf. [WGCI09]).

Binding Mechanism	Code Sample	Description

Bind to variable	// Update b when a changes
	`var b = bind a;`
	
	// Update z when x or y changes
	`var z = bind x + y;`
		Update the value of a bound variable, whenever a dependent variable from the binding context is changed.

Bind to instance variable	// Update y when x changes
	`var m = myClass`
	`{ y: bind x, z: x };`
		The value of x is assigned to both instance variables y and z. A change of x updates y but not z.

Bind to condition expression	// Update z corresponding to condition when x or y changes.
	`var z = bind if (x < y)`
	`x else y`
		Whenever the variables of the condition expression changes, the condition exp. is re-evaluated and the value of the appropriate branch variable is assigned to `z`.

Bind to for expression	// Constant `a` is subject to update whenever `b`, `c` or `k` changes
	`def a = bind for (i in`
	`[b..c] where i < k) {}`
		Changes of all variables from the `in` clause and the `where` clause expect incremental variable cause reassignment of bound variable.

Bind to a block	// Update z when `a` or `b` changes, not `c`.
	`def z = bind {`
	`  def y = a;`
	`  def x = b;`
	`  def w = c;`
	`  y + x; }`
		Bound blocks can only contain constant declarations and a final value expression stating the return value. Updates occur when sources of the return value expression change.

Bind to object literal	// Create a new instance of `myClass` and assign it to `m` whenever `a` or `b` changes.
	`def m = bind myClass`
	`{ x:a, y:b };`
		If the value of an instance variable initializer of the bound object literal expression changes, a new instance is created. If one of the instance

	// Similar as above. A change in `a`, however, leads only to a change in the property `n.x`. `def n = bind myClass {x: bind a, y: b };`	variable initializers is additionally bound to a variable, only the property of the instance receives an update.

Bidirectional and lazy binding	// Update `x` when `y` changes and vice versa.
	`var x = bind y with reverse;`
	// Update `m` as soon as it is accessed with all updates occurred (queued) after the last access-triggered "batch" update.
	`var m = bind lazy n + 5`
	`println( "m: " {m} )`
		Enhance default binding behavior. Bidirectional binding propagates changes in both the target object and the dependent objects to each other.
		Lazy binding has a performance impact. The target object will not be updated before it is accessed. The change in the dependent object is registered, though.

Bind to function call	// Update `rating` with the new result of `calcRating` when `a` or `b` changes. `var rating = calcRating(a, b, c)`	Whenever the value of an argument of the bound function changes, the function is invoked again and the result is propagated to the target variable.

Bound function	// Update `rating` with the new result of `calcRating` when `a`, `b` or `z` changes.
	`bound function calcRating(x: Number, y: Number) {`
	`  var m = x;`
	`  var n = y;`
	`  var o = z;`
	`  var rating = calcRating(a, b)`
	`}`
		A bound function is a function definition decorated with the `bound` modifier [WGCI09]. In contrast to binding a function call, a bound function will also be re-invoked when the function body changes.

Table 7: JavaFX Binding Capabilites

Please note that you can only bind variables on declaration not on a subsequent initialization and once bound the variable can only be reassigned in the course of updates from bidirectional binding.

Triggers

Triggers are designed to catch data modification events on an observed variable. Attach a trigger to a variable with the keywords `on replace` followed by an optional variable for storing the old value and a block which should be executed every time the value of the variable changes (cf. listing 39).

```
var x: String on replace oldValue {
    y = x;
}
var y: String;
```

Listing 39: JavaFX Trigger Example

A trigger can be used to mimic the behavior of a binding. It actually bypasses the restriction of reassigning bound variables. However, triggers only listen for changes on the observed variable. They do not observe synchronization of target and source.

6.3 Remote Service Handling

The same principles as already discussed on the basis of other frameworks effect service handling in JavaFX. A mechanism is required in order to asynchronously make a request to a provided URL. As soon as the service returns a response, another mechanism is required for parsing the response and integrating the data in your application.

Before version 1.2, released on June 2 2009, JavaFX used to leverage `javafx.async.RemoteTextDocument` for server communication. Since version 1.2 however, `RemoteTextDocument` is deprecated as one is encouraged to use `javafx.io.http.HttpRequest` instead. `HttpRequest` breaks down the communication process into a series of steps. Each step can have a callback assigned and allows inference to be drawn about the state of transmission. Use `javafx.data.pull.PullParser` to handle XML or JSON based responses.

The concrete example displayed in figure 35 with its code depicted in the listings 40 to 42 is based on a REST example from [CCB09]. Contrary to the referenced example, the following one is implemented with `HttpRequest` instead of the deprecated `RemoteTextDocument`.

When looking at figure 35 one can expect an application consuming a weather web service. In fact, the application consumes a service from the Yahoo API[53] first and then calls a weather service from the GeoNames API[54] on the basis of the retrieved coordinates as depicted in listing 40.

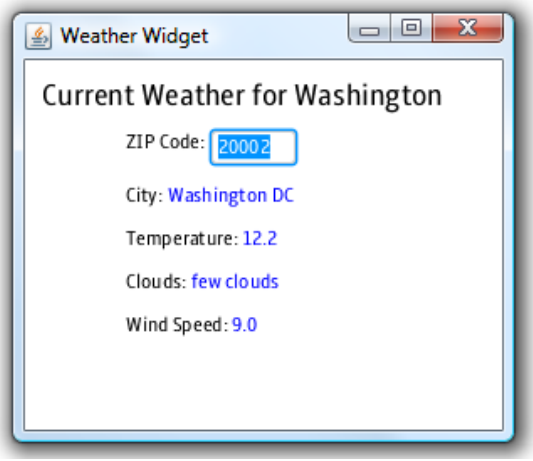


Figure 35: JavaFX Weather Widget

The very first step[55] as depicted in listing 40 is to create an `HttpRequest` instance, specify the request method (GET or POST) and the URL (`location`) and assign a callback for parsing the response to the `onInput` instance variable initializer (1). Specifying additional callbacks allows for fine-grained state tracking of the operation, e.g. assign another callback to `onStarted` to record the request initialization or assign a callback to `onDone` that is called when the request is completed. Next the `start()` function needs to be invoked on the request to execute the Yahoo service request (2).

If input arrives with the response of the service, the `parseLocationResponse()` function previously assigned to `onInput` is called (3). It instantiates the `PullParser` class with the respective document type (XML or JSON), the input stream and an `onEvent()` function invoked each time a value is fully parsed. Calling the `parse()` method on the `PullParser` object starts the parsing process (4). The `onEvent()`

[53] http://developer.yahoo.com/everything.html – requested in March 2011
[54] http://www.geonames.org/export/web-services.html – requested in March 2011
[55] See http://download.oracle.com/docs/cd/E17802_01/javafx/javafx/1.3/docs/api/javafx.io.http/javafx.io.http.HttpRequest.html – requested in March 2011 – for the documentation of the `javafx.io.http.HttpRequest` class and its callback variables.

function compares the fully qualified name of each element with those of interest and stores the element's value in a respective instance variable of an object in case of a positive match (5).

For instance, the instance of a class like

```
public class Location {
    public var city: String;
    public var state: String;
    public var lat: String;
    public var long: String;
}
```

can be used to store the data. As soon as the location data is retrieved, parsed and stored (6), a new request is initiated on the basis of the data calling the GeoNames service for weather information. The request and the parsing function are designed analogous to the location counterparts (following the steps (1) to (6)).

```
var location: Location = Location {};
var weather: Weather = Weather {};
var zipCode: String;
var locationURL = bind "http://local.yahooapis.com/LocalSearchService/"
                       "V3/localSearch?appid=YahooDemo&query=city&zip={zipCode}"
                       "&results=1&output=xml";
var weatherURL = bind
            "http://ws.geonames.org/findNearByWeatherXML?lat={location.lat}"
            "&lng={location.long}";
var req: HttpRequest;
var locationParser: PullParser;
var weatherParser: PullParser;

function requestCoordinates(): Void {
  req = HttpRequest {
              method: HttpRequest.GET
              location: bind locationURL
              onInput: parseLocationResponse
              onDone: function() {
                  println("Coordinates retrieved.");
                  requestWeather()
              }
          }
  req.start();

  println("Coordinates requested.");
}

function parseLocationResponse(is: InputStream): Void {
  try {
      locationParser = PullParser {
         documentType: PullParser.XML
         input: is
```

1

2

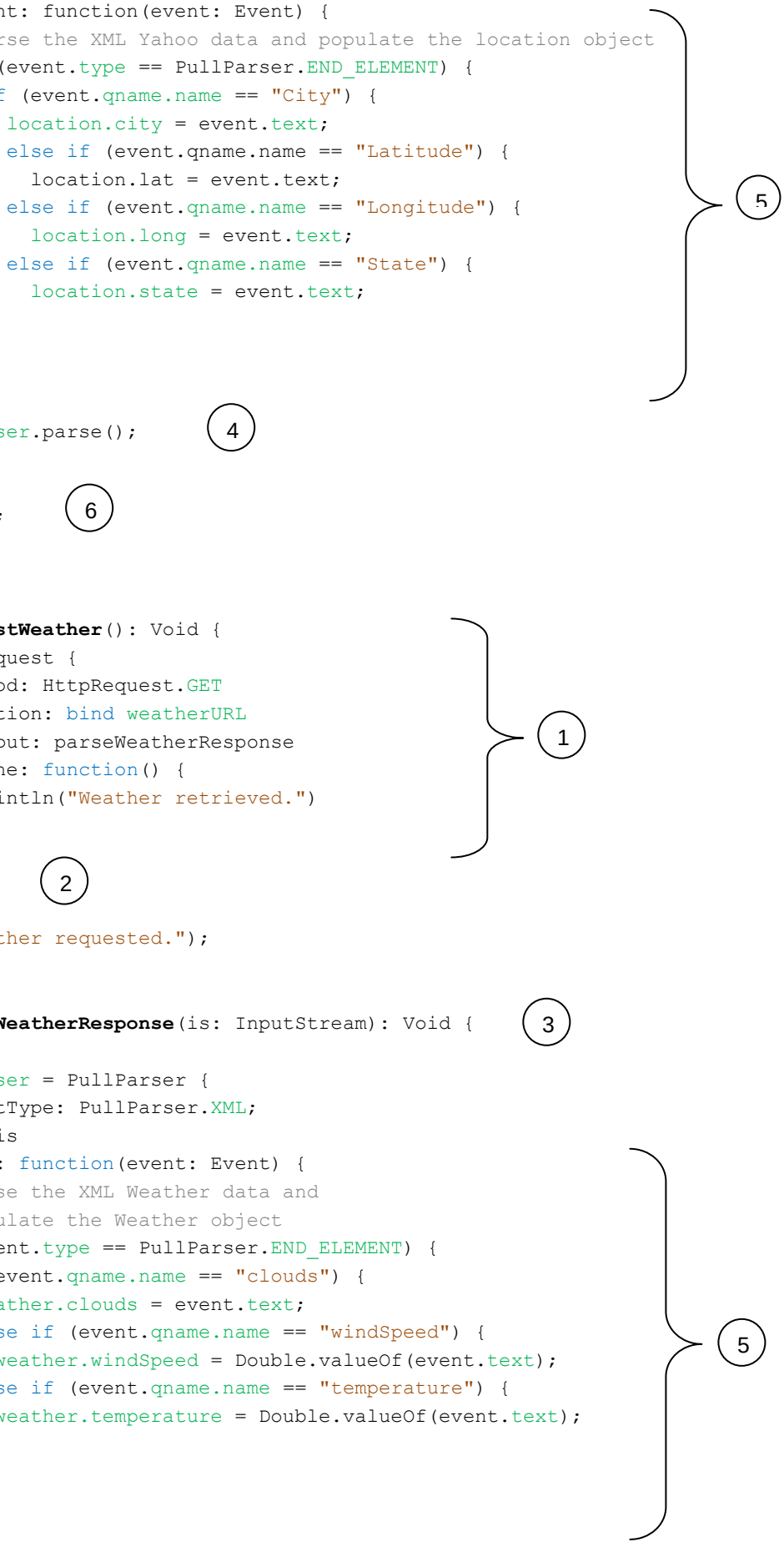
3

```
          onEvent: function(event: Event) {
          // parse the XML Yahoo data and populate the location object
            if (event.type == PullParser.END_ELEMENT) {
              if (event.qname.name == "City") {
                location.city = event.text;
              } else if (event.qname.name == "Latitude") {
                  location.lat = event.text;
              } else if (event.qname.name == "Longitude") {
                  location.long = event.text;
              } else if (event.qname.name == "State") {
                  location.state = event.text;
              }
             }
          }
       }
    locationParser.parse();
  }
  finally {
    is.close();
  }+
}

function requestWeather(): Void {
  req = HttpRequest {
          method: HttpRequest.GET
          location: bind weatherURL
          onInput: parseWeatherResponse
          onDone: function() {
            println("Weather retrieved.")
          }
  }
  req.start();

  println("Weather requested.");
}

function parseWeatherResponse(is: InputStream): Void {
  try {
    weatherParser = PullParser {
       documentType: PullParser.XML;
       input: is
       onEvent: function(event: Event) {
        // Parse the XML Weather data and
        // populate the Weather object
        if (event.type == PullParser.END_ELEMENT) {
          if (event.qname.name == "clouds") {
            weather.clouds = event.text;
          } else if (event.qname.name == "windSpeed") {
              weather.windSpeed = Double.valueOf(event.text);
          } else if (event.qname.name == "temperature") {
              weather.temperature = Double.valueOf(event.text);
          }
        }
      }
    }
```



```
    weatherParser.parse();          (4)
  } finally {
      is.close();                   (6)
  }
}
```

Listing 40: JavaFX Weather Widget Logic

As an aside, the code from listing 40 demonstrates the JavaFX Script specific way of string concatenation. JavaFX Script basically requires no concatenation operator to join multi-line strings as adjacent string literals are automatically merged. In order concatenate a string literal with a variable forming a string expression, JavaFX Script uses curly braces "{}"; in contrast to Java which uses the "+" operator. For instance, the `locationURL` variable has the concatenation of a string literal plus the brace-delimited value of the `zipCode` variable assigned (bound) to it:

```
var locationURL = bind "http://local.yahooapis.com/LocalSearchService/"
                       "V3/localSearch?appid=YahooDemo&query=city&zip={zipCode}"
                       "&results=1&output=xml";
```

A workaround would include the `concat()` function:

```
var locationURL = bind "…YahooDemo&query=city&zip=".concat(zipCode).concat("…")
```

The last step is to integrate the application logic with your GUI. Create a container like an `HBox` with a `TextBox` which accepts ZIP code input from the user – the basis of the search results (cf. listing 41).

```
HBox { content: [
    Text {
      content: "ZIP Code: "
    },
    TextBox {
      columns: 6
      text: bind zipCode with inverse
      selectOnFocus: true
      action: function(): Void {
      requestCoordinates();
    }
  }]
}
```

Listing 41: JavaFX Weather Widget ZIP Request View

Hitting the `RETURN` key calls the `requestCoordinates()` function which initiates the `HttpRequest` to Yahoo. Note the bidirectional binding expression assigned to the `text` instance variable initializer of the `TextBox` object. Every time the `text` property changes, the `zipCode` is also changed and vice versa to ensure that requests are only based on the current `zipCode`. Create further visual containers with text boxes to display the retrieved search results in a structured way like in listing 42:

```
HBox { content: [
    Text {
     textOrigin: TextOrigin.TOP
     content: "City: "
    },
    Text {
      textOrigin: TextOrigin.TOP
      content: bind "{location.city} {location.state}"
      fill: Color.BLUE
  }]
}
```

Listing 42: JavaFX Weather Widget Response View

6.4 Interaction with the Environment

JavaFX classes are compiled to Java bytecode and executed on the JVM. So JavaFX runs entirely on the Java platform. JavaFX applets have to be deployed on HTML pages. So JavaFX applets have to be integrated via JavaScript. Now these prerequisites actually unlock great interaction potentials between JavaFX and Java and JavaScript respectively.

6.4.1 Interaction with JavaScript

The functionality of the `AppletStageExtension` class is not limited to defining the behavior of draggable applets but also provides the `eval()` function for access to JavaScript functions and DOM. `eval()` takes any JavaScript expression as a string for evaluation. For instance, assign the instance of `AppletStageExtension` to a variable `applet` and then invoke:

```
applet.eval("someJsFunc()");
```

to execute the JavaScript function on the JavaFX side and get a possible return value.

Calling JavaScript FX functions and passing data from JavaScript to JavaFX is more restricted. As of JavaFX 1.3.1, released on August 21 2010, only public script variables and functions defined for the applet can be accessed directly. Nevertheless, any script level function can be accessed via the keyword `document.appletID.script` which exposes the `script` property of the applet - instantiated on the web page and referenced by its `id` – to the JavaScript engine.

Now call a JavaFX script function like:

```
document.appletID.script.javaFXFunc(args);
```

In order to ensure that function invocation does not interfere with ongoing tasks, `javafx.lang.FX.deferAction()` can be used to brace the body of the function JavaScript may calls. This makes sure that the task is queued for execution after the current event is terminated[56]. Listing 43 shows a simple JavaFX nutshell with a script level function `setColor()` which changes the value of the `mycolor` variable. When the nutshell is run a single circle is drawn on the screen filled with a color determined by the `mycolor` variable.

```
var mycolor = Color.GOLD;
public function setColor(red: Number, green: Number, blue: Number): Void
{
    mycolor = Color { red: red, green: green, blue: blue };
}
public function run() {
    Stage {
        scene: Scene {
            content: [
                Circle {
                    centerX: 200
                    centerY: 200
                    radius: 80
                    fill: bind mycolor
                }
            ]
        }
    }
}
```

Listing 43: JavaFX Script Function Set Up for Invocation in JavaScript

[56] cf. the `invokeLater()` method of the `javax.swing.SwingUtilities` class http://download.oracle.com/javase/6/docs/api/javax/swing/SwingUtilities.html#invokeLater%28java.lang.Runnable%29 – requested in April 2011

Listing 44 basically depicts the same code as listing 37 necessary to initialize an applet. However, it also assigns an `id` to the applet. The JavaScript function `changeColor()` gets the applet by its id and invokes its `setColor()` function previously defined in JavaFX. As a consequence, the `mycolor` script level variable of the JavaFX applet is changed.

```
<script src="http://d.javafx.com/1.3/dtfx.js"> </script>
<script>
    javafx(
        {
          archive: "JavaFXDeploymentTest.jar",
          draggable: true,
          width: 200,
          height: 200,
          code: "javafxdeploymenttest.Main",
          name: "JavaFXDeploymentTest"
          id: "JavaFXApp"
        }
    );
    function changeColor() {
        var app = document.getElementById("JavaFXApp");
        app.script.setColor(1.0, 1.0, 0.0);
}

</script>
```

Listing 44: Invoking JavaFX Script Function in JavaScript

This bridge can be used to provide access to JavaFX and Java packages already imported in the applet context. As of JavaFX 1.3.1 it is not possible to instantiate JavaFX classes in JavaScript. This restriction does not apply to Java classes, though.

For example, create an instance of a Java class imported in the JavaFX applet:

```
var dim = new document.appletID.java.awt.dimension();
```

and assign a value to one of its properties:

```
dim.height = 200;
```

6.4.2 Interaction with Java

With JavaFX one can perform tasks like GUI building and model/view data synchronization significantly more efficient than with plain old Java. Basically, scripting languages tend to perform specific tasks more easily and more quickly than general-purpose languages. Think of implicit type declaration of variables and automatic type conversions. In times when you occasionally reach the limits of the JavaFX API you can take advantage of the vast number of Java libraries available to JavaFX.

But it does not have to be the need of certain functionality that encourages one to integrate Java classes in a JavaFX application. Image a scenario with a front end written in JavaFX connecting to a Java back end employing strongly typed data communication.

6.4.2.1 Basic Java Integration

Any class from the Java API can be leveraged inside JavaFX without any adjustments required. Just use the `new` operator to create an instance and use the “dot notation” (.) to access variables and functions of instance objects. For instance

```
var mySwingLabel = new javax.swing.JLabel();
```

creates an instance of the class `javax.swing.JLabel` in JavaFX. In order to assign a value to the instance’s `text` property, invoke the `setText()` method on it:

```
mySwingLabel.setText("I'm a swing label.");
```

So working with Java classes in JavaFX is basically the same as in Java itself. Just observe name conflicts with JavaFX keywords like `insert` and `delete` when accessing properties of a Java object E.g. calling `java.lang.System.in.read();` results in a compilation error as `in` is a reserved keyword [CCB09]. To avoid this quote the field: `java.lang.System.<<in>>.read();` .

A JavaFX class does not differentiate between extending Java classes and implementing Java interfaces; it extends both and overrides their methods. In fact, a Java interface is treated as a mixin class (supports multi inheritance).

Listing 45 demonstrates how a JavaFX class extends (= implements) the Java interface `java.awt.event.ActionListener`. It has to override all available abstract methods of the interface.

```
class MyActionListener extends java.awt.event.ActionListener {
    public override function
        actionPerformed(e:java.awt.event.ActionEvent) : Void {
            // handle event
    }
}
```

Listing 45: JavaFX Class Extending Java Interface

Although it is possible to extend a Java class in JavaFX, its attributes and methods have no inherit connection to binding and triggering mechanism; the runtime system will ignore the updates [CCB09]. The main obstacle is Java lacks facilities to track changes made to fields[57].

JavaFX arguments of primitive data type are autoboxed into appropriate Java wrapper classes when passed on Java method call.

Table 8 maps all JavaFX primitive data types[58] to their corresponding Java classes[59]. JavaFX literals are also JavaFX objects, so it is possible to access the respective methods of a class directly from the literal [CCB09]. For instance, a double literal like 2.25 can be converted to an integer by calling its `intValue()` method inherited from `java.lang.Double`.

[57] For further reading on binding Java objects in JavaFX Script and a workaround to accomplish this see http://blog.netopyr.com/2008/12/19/binding-java-objects-in-javafx-script/ – requested in May 2011

[58] Technically “String” is no primitive data type but, due to its special support for character strings, often considered as such.

[59] For a full list of conversion rules in cases when automatic conversion is not possible see [CCB09] pp. 286-291.

JavaFX Type	Java Class
Double	java.lang.Double
Number	java.lang.Number
Float	java.lang.Float
Long	java.lang.Long
Integer	java.lang.Integer
Short	java.lang.Short
Byte	java.lang.Byte
Character	java.lang.Character
Boolean	java.lang.Boolean
String	java.lang.String

Table 8: JavaFX to Java Type Mapping

JavaFX 1.2, released on June 2 2009, introduced the `native array of` type to share arrays between JavaFX and Java. JavaFX's basic collection data type `sequence` is not treated as array and incompatible to native Java arrays. Whenever a Java array parameter is required or Java returns an array, the `native array of` type is used to mimic a Java array.

6.4.2.2 Integration of JavaFX in Java

Accessing JavaFX objects values from within Java is more complicated than the other way round. The main obstacle is the fact, that the Java compiler is not able to read JavaFX scripts directly to extract the needed information [Hein08]. Fortunately, various approaches exist for scripting JavaFX in Java. You can call JavaFX functions via Java interface, execute scripts with the FXEvaluator, leverage the Java Scripting API or use Reflection to "investigate" JavaFX objects at runtime.

6.4.2.2.1 Java Interfaces

The simplest approach is to define a Java interface like in listing 46 and have a JavaFX class extending it and overriding (all of) its method(s)[60].

```
public interface Printable {

    void printMessage();
}
```

Listing 46: Java Interface for Implementation in JavaFX

The next step is to create a Java class with a method (`print()`), which takes an instance of the JavaFX class (represented by `m` of type `Printable`) and invokes the implemented interface method (`printMessage()`) on it as depicted in listing 47.

```
public class JavaMessagePrinterLibrary {

    public static void print(Printable m) {
        m.printMessage();
    }
}
```

Listing 47: Java Class Receiving JavaFX Instance and Invoking Method on It

The last step is to instantiate the JavaFX class in JavaFX and to pass the instance (`jFxObj`) to Java by calling the appropriate method (`print()`) from the Java class[61] demonstrated in listing 48.

```
public class JavaFXMessagePrinter extends Printable {
    public var message: String;
    public override function printMessage() {
        System.out.println(message);
    }
}
```

[60] cf. http://java.sun.com/developer/technicalArticles/scripting/javafx/javafx_and_java/index.html – requested in March 2011

[61] Ensure that the JavaFX class and the Java class are in the same package. As a consequence, the Java class can be referenced as such in JavaFX.

```
var jFxObj = MyJavaFXClass {
             message: "Hello JavaFX from Java by Interface!"
         };
function run(args: String[]): Void {
    JavaMessagePrinterLibrary.print(jFxObj);
}
```

Listing 48: JavaFX Class Implementing Java Interface

The advantage is that the Java class does not need to know that it is invoking JavaFX functionality. Neither the Java class nor the Java interface contains JavaFX code. Hence, they are reusable along with other Java classes.

6.4.2.2.2 FXEvaluator

The `javafx.util.FXEvaluator` class should be used to execute JavaFX scripts from within Java. It requires having `javafxc.jar`, which represents the JavaFX Script compiler API, added to the classpath. The `eval()` method of `FXEvaluator` takes a string representation of the script, executes the script and returns a JavaFX object, if the script creates one. The following line runs the script and assigns the result to an object:

```
Object fxObj = FXEvaluator.eval(script);
```

Please note that the returned JavaFX object can only be used to inform about the type of class not to access properties of it. The script is evaluated without any context state (script context). Any state it creates during evaluation cannot be reused by other scripts[62]. `FXEvaluator` is suitable for simply executing scripts but not to pass in arguments to a script. This requires working with the Java Scripting API.

6.4.2.2.3 Java Scripting API

Scripting in Java is supported by two frameworks: Scripting for the Java Platform (JSR 223) and the Bean Scripting Framework (BSF). Both allow accessing Java objects and methods from scripting languages and executing programs written in scripting languages in Java applications.

---

[62] cf. http://download.oracle.com/docs/cd/E17802_01/javafx/javafx/1.3/docs/api/javafx.util/javafx.util.FXEvaluator.html – requested in March 2011

Scripting for the Java Platform

Scripting for the Java Platform (often referred to as JSR 223) defines mechanisms to access scripting languages from Java code. An implementation for JavaFX Script is available and part of the JavaFX Runtime. With the provided scripting engine, you can read JavaFX files, compile and run them. And because the engine for JavaFX Script implements the interface `Invocable`, it is possible to call members of JavaFX objects [Hein08].

Version 6 of the Java Standard Edition, released on December 11 2006, implemented JSR 223, which enables scripting on the Java Platform with any scripting engine as long it is JSR 223 compatible. The API is located in `javax.script` and is distributed with every Java 6 SE. The basic steps for running a script in Java are[63]:

- Create a `ScriptEngineManager` instance:

  ```
  ScripEngineManager manager = new ScriptEngineManager();
  ```

  The `ScriptEngineManager` is used to discover available script engines and instantiates them.

- Retrieve a ScriptEngine instance from the ScriptEngineManager:

  ```
  ScriptEngine scrEng = manager.getEngineByExtension("javafx");
  ```

  Representation of the specific script engine; also possible to retrieve by invoking `getEngineByName()` on `ScriptEngineManager` object.

- Evaluate (execute) script by calling `eval()` on the script engine instance.

The script engine instance can be optionally cast to a `JavaFXScriptEngine`. As a consequence, the script can be compiled before evaluation and a `javax.tools.DiagnosticCollector` object, collecting errors in case of a `ScriptException`, can be passed to the `eval()` or `compile()` methods of the `JavaFXScriptEngine` instance.

So far working with the Java Scripting API seems to be similar to `FXEvaluator`. A script engine, however, can be used to call methods and to assign values to instance variables from previously executed scripts. Use `invokeMethod(fxObj,`

[63] cf. http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/ – requested in March 2011

`"methName")` on the engine instance to call a method. `fxObj()` stands for the object the evaluated script returned; `methName` is to be substituted with the name of the appropriate method. Calling `put(key, value)` places a key/value in the state of the engine. This can be basically used to assign a value to a variable of the script instance on execution.

Bean Scripting Framework

BSF[64] refers to a scripting API originally developed by IBM and now maintained by the Apache Software Foundation. BSF has basically a very similar range of application as `javax.script` (JSR223). The `BSFManager` class handles all supported scripting engines and gives access to scripting services. Invoke `loadScriptingEngine()` on the `BSFManager` instance to get an object implementing a `BSFEngine` which reflects the scripting capabilities of a certain language and permits script execution. Currently, two different version of BSF exist: the BSF 2.x releases (BSF 2.4 was released on October 6 2006) build on the original API by IBM and the BSF 3.x releases implement `javax.script` (JSR223) for Java 1.4 and Java 1.5.

Scripting Synergy in Action

The following nutshell demonstrates how two scripting languages achieve to communicate with each other in Java. The two script languages are JavaFX (for obvious reasons) leveraging the JSR 223 API and ooRexx leveraging BSF.

Open Object Rexx[65] (ooRexx) is a free implementation of Object Rexx managed by Rexx Language Association (RexxLA). ooRexx is a fully object-oriented, interpreted language. It allows weak typing, imposes few syntax rules and consists of commands full of meaningful English words. Unlike languages like Java or C#, ooRexx uses the tilde character to invoke methods on objects (send messages to the object) instead of a dot. Apart from its powerful functionality, ooRexx was chosen for this demonstration because it is supported with its own BSF engine: BSF4ooRexx[66, 67].

---

[64] http://jakarta.apache.org/bsf/ – requested in March 2011
[65] http://www.oorexx.org/ – requested in March 2011
[66] http://wi.wu-wien.ac.at/rgf/rexx/bsf4oorexx/current/ – requested in March 2011
[67] A thesis published in July 2010 introduces a JSR 223 bridge for ooRexx. See http://wi.wu-wien.ac.at:8002/rgf/diplomarbeiten/BakkStuff/2010/201007_Ryabenkiy/201007_Ryabenkiy_WebScripting_ApacheTomCat_TagLib.pdf for more information.

Listing 49 depicts the Java nutshell facilitating the interfacing from JavaFX to ooRexx on the basis of two simple scripts. The JavaFX script is based on a sample from [WGCI09] while the ooRexx script is based on samples from Rony G. Flatscher. Successfully running the demonstration nutshell requires adding `javafxc.jar`, `bsf-rexx-engine.jar` and `bsf-v40090910.jar` to the application's classpath.

The Java nutshell executes a JavaFX script, catches the returned object and invokes a JavaFX function on the object. The JavaFX instance returned in the course of the script evaluation is subsequently passed to an ooRexx script. On execution of that script, ooRexx sends messages to the passed JavaFX object.

```
import javax.script.*;
import com.sun.javafx.api.JavaFXScriptEngine;
import org.apache.bsf.*;

public class Main {

  public static Object result;
  public static Object javaFXObj;
  public static void main(String[] args) {
    String javaFxCode =
            // Create JavaFX class using object literal syntax
            "public class Student {\n"
            + " public var name:String;\n"
            + " public function getName():String { \n"
            + " this.name;\n"
            + " }\n"
            + "}\n"
            + " function run(args: String[]):Void { \n"
            + " var student = Student { name: 'John Dorian'\n"
            + " as String};\n"
            // JavaFX built-in function
            + " println('[JavaFX] Name = {student.name}');\n"
            + " student;\n"
            + "}";

    // Get manager for script engine discovery
    ScriptEngineManager manager = new ScriptEngineManager();

    // Lookup script engine and create instance
    ScriptEngine scrEng = manager.getEngineByExtension("javafx");
    // Cast script engine to JavaFXScriptEngine which allows invocation of
    // procedures in script.
    JavaFXScriptEngine engine = (JavaFXScriptEngine) scrEng;
```

```
if (engine == null) {
    System.out.println("no scripting engine available");
} else {

    try {
        fxObj = engine.eval(javaFxCode);    // Execute script
        try {
           // Print string representation of JavaFX object
           System.out.println("[Java] javaFXObj: " + javaFXObj);
           // Get the class of JavaFX object and print it
           System.out.println("[Java] javaFXObj.getClass: " +
                              javaFXObj.getClass());
            // Call method on script object
            result = engine.invokeMethod(fxObj, "getName");
            // Print result of method invocation
            System.out.println("[Java] getName() returned: " + result);
        } catch (NoSuchMethodException nsme) {
            nsme.printStackTrace();
        }
    } catch (ScriptException ex) {
        ex.printStackTrace();
    }
}
try {
    // Similar to ScriptEngineManager; implements discovery
    BSFManager mgr = new BSFManager();

    // Lookup script engine and create instance
    BSFEngine rexxEngine = mgr.loadScriptingEngine("rexx");
    // Rexx code to run; triggering the requires-processing for "BSF.CLS"
    rexxEngine.apply("", 0, 0, "::requires BSF.CLS;", null, null);
    // ooRexx script code
    // retrieve the beanName (index into BSFRegistry)
    String rexxCode = "use arg javaFXObj ;" +
                // send Java object a message; get string representation
                // of JavaFX object
                "say '[ooRexx] javaFXObj:' javaFXObj;" +
                // send Java object another message; get class type
                "say '[ooRexx] javaFXObj~class: ' javaFXObj~class;" +
                // send Java object another message; invoke getName() and
                // display the result reversed
                "say '[ooRexx] javaFXObj~getName~reverse: '" +
                "javaFXObj~getName~reverse;" +
                // get ooRexx support (camouflage Java as ooRexx)
                 "::requires BSF.CLS ;";
    // argument for ooRexx
    Vector vArgs = new Vector();
    // include JavaFX return value in argument for ooRexx
    vArgs.addElement(javaFXObject);
```

```
            // Execute script code; first arg: source*, second arg: line number*,
            // third arg: column number*, fourth arg: script string,
            // fifth arg: optional Java args names, sixth arg: Java args
            // *relevant in case of external file source
            rexxEngine.apply("", 0, 0, rexxCode, null, vArgs);
        } catch (BSFException e) {
            e.printStackTrace();
        }
    }
}
```

Listing 49: JavaFX-Java-ooRexx Interaction via JSR 223 and BSF 2.4

Figure 36 shows the output of the nutshell from listing 49. The first output line is printed by JavaFX. The second line is the JavaFX object the script evaluation returned printed by Java. The class of the JavaFX object as recognized in Java is displayed in line three while line four contains the value the JavaFX function invocation returned. The fifth, sixth and seventh lines are generated by ooRexx. In line five ooRexx displays the string representation of the passed JavaFX object. In line six the proxy class of the JavaFX object is shown. In the last line ooRexx sent a message to the JavaFX object to call its `getName()` function and to invoke `reverse()` on it.

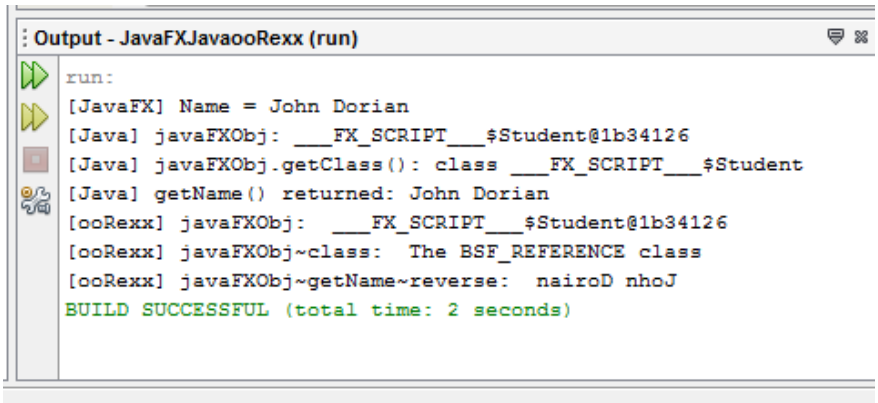


Figure 36: Output of JavaFX-Java-ooRexx Interaction Nutshell

6.4.2.2.4 JavaFX Script Reflection

JavaFX Script includes the reflection API `javafx.reflect` that allows you to perform certain metaprogramming tasks. Metaprogramming is the act of manipulating a programming facility using a language rather than using the programming facility itself [WGCI09].

The Reflection API makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time. It is also possible to instantiate new objects, invoke methods and get/set field values using reflection [JenkJR].

Using the Reflection API is by far the most challenging approach for processing a JavaFX script in Java. It is all about working with proxies so-called mirrors of objects and values in a given context.

The entry point is the class `javafx.reflect.FXLocal.Context` retrieved via

```
FXLocal.Context context = FXLocal.getContext();
```

Whenever you need to work with an object or value, work with a reference of it created via `context.mirrorOf(obj)`. This function wrappers the value with a proxy that uses the local VM to handle the reflection [CCB09]. Table 9 depicts vital JavaFX reflection functionality.

Task	Code Sample

Create instance and initialize instance variables	/* Get reference of class (from JavaFX API or custom class) which is going to be instantiated. */
	`FXClassType classRef = context.findClass("javafx.geometry.Dimesion2D");`
	
	// Get raw uninitialized object (allocate memory).
	`FXLocal.ObjectValue obj = (ObjectValue) classRef.allocate();`
	
	/* Initialize instance variables with the name of the variable and a proxy of the value as arguments. */
	`obj.initVar("height", context.mirrorOf(10.0));`
	`obj.initVar("width", context.mirrorOf(20.0));`
	
	`obj.initialize();` // Initialize instance of class.

Assign value(s) to instance variable(s)	// Previously created instance of
	// class `javafx.geometry.Dimension2D`
	var dim = Dimension2D { height: 10.0, width: 10.0 };
	
	// Get reference to class (cf. row above)
	`FXClassType classRef = context.findClass("javafx.geometry.Dimension2D");`

	// Get a proxy of the instance object “dim”
	`FXObjectValue obj = context.mirrorOf(dim);`
	
	// Access instance variables of referenced class with a given name
	`FXVarMember hVar =`
	`classRef.getVariable("height");`
	`FXVarMember wVar =`
	`classRef.getVariable("width");`
	
	/* Assign value to variable with the specific instance object and a proxy of the value as argument. */
	`hVar.setValue(obj, context.mirrorOf(10.0));`
	`wVar.setValue(obj, context.mirrorOf(20.0));`

Assign value(s) to script variables(s)	/* As already mentioned, script variables are the almost JavaFX equivalent to Java static fields. The workflow is basically the same as with instance variables with one exception: “null” has to be passed to `setValue()` instead of the object instance. */
	
	// Previously declared script variable
	`public var s: String;`
	
	// Get reference to class
	`FXClassType classRef =`
	`context.findClass("MyJavaFXClass");`
	
	// Access script variable of referenced class with a given name
	`FXVarMember hVar =`
	`classRef.getVariable("s");`
	
	/* Assign value to variable with “null” as first parameter instead of an object instance (there isn’t any) and a proxy of the value as argument. */
	`hVar.setValue(null,`
	`context.mirrorOf("I’m a string."));`

Invoke function	/* Get function of referenced class that matches the passed name. Also pass arguments if function takes any. */
	`FXFunctionMember func =`
	`classRef.getFunction("toString");`
	
	/* Invoke function with instance object and value proxies if function takes arguments. */
	`FXValue val = func.invoke(obj);`

Table 9: JavaFX Reflection Use Cases (cf. [CCB09] and [WGCI09])

7 Evaluation

The previous four chapters presented the big players in RIA development: AJAX, Adobe Flex, Microsoft Silverlight and JavaFX; and discussed selected functionality they offer. The goal of this chapter is to highlight the strengths and weakness of each technology. In addition, use cases are presented to help spotting the technology which meets your requirements the most. Please note that these guidelines are generalized. The most significant influencing factors are programming and tool expertise of the staff, the compatibility with legacy systems and the philosophy of an enterprise.

7.1 Evaluation of AJAX

Working with AJAX is a mixed blessing. On the one hand, developers profit from the wide spread of the utilized standards and can quickly get productive. On the other hand, a lot of time and cost has to be spent on crucial compatibility tests for heterogeneous platforms and versions. Workarounds have to be set up too, if users deactivate JavaScript in their browsers.

Strengths

- Standardized technologies:

  AJAX leverages open standardized technologies, which are available in almost every browser. Building on Web standards like HTML, CSS and JavaScript basically ensures forward- and backward-compatibility, better maintenance and reaching a broad audience.

- Open Source Frameworks and Community:

  Various open source frameworks supporting AJAX exist on the Web (cf. table 3). They do not only simplify the development of AJAX applications but also provide custom smart controls and other building blocks. Users of such open source software have access to the source code and debugging tools and may suggest both bug fixes and enhancements to the source code. Open source software users contribute because they are interested in improving the reliability and maintainability of the software. The successful

adoption of open and interoperable AJAX-based Web technologies is also pursued by the OpenAjax Alliance[68], a 100+ member organization of leading vendors, open source projects, and companies using AJAX.

- Familiar look and feel:

  Basic controls employed in AJAX applications like buttons, text input fields, radio buttons, checkboxes, etc. are exactly the ones web users are familiar with and intuitively use them.

- Favors gradual re-factoring of existing Web applications:

  Introducing AJAX has minimal (negative) impact on productivity. AJAX widgets do not require a holistic changeover as long existing applications are written in HTML or JavaScript. What is more, common web developers should be familiar with the languages of the AJAX conglomerate since they are standards for many years.

- Light footprint:

  Clients may reject to install a plug-in in order to use an application. The initial state of an AJAX application solely requires downloading HTML/script files and CSS files without any extra gadgets to install.

- Development tool independence:

  It is basically a benefit when one does not have to rely on a single (commercial) IDE to develop an application. Sophisticated free IDEs like NetBeans and Eclipse come with plug-ins simplifying the design process.

- No updates on the client required:

  When users aren't on the latest version of a plug-in, requiring them to update is risky. According to [Hamm06] many are unwilling to wait through the lengthy install process and instead choose to do business elsewhere – potentially with the competition.

[68] http://www.openajax.org/index.php – requested in May 2011

- SEO:

  Search Engine Optimization (SEO) aims to improve the visibility of web sites to search engines and their users. Indexing dynamic Web content and the content of RIAs may be a blind spot. Although Adobe is providing a special type of the Flash Player to Google and Yahoo! for prospecting information in `.swf` files for a short time, AJAX applications are still the most transparent ones. For instance, hash fragments (#) in URLs can be used. Hash fragments (#) are generally used in an URL to identify a position within a resource. In JavaScript the `window.location.hash` property can be used to get and set the application's state. For instance, in a web mapping service the value of the hash fragment can be used to reflect certain zoom levels. The `onhashchange` event allows detecting when the hash changed and the user navigated on the map. In combination with an appropriate event handler, this function can be used to update the address bar every time an AJAX action occurred. As a consequence, the hash fragment can be used to record and directly link the application's state while the search engine will index the original page that should be ranked. In addition, Google which is embracing AJAX for a long time (think of Gmail and Google Maps) pushes AJAX SEO and offers guides for making AJAX application crawlable[69].

Weaknesses

- Complex cross browser behavior and accessibility:

  The existence of various web browsers makes the platform very heterogeneous. Not only handling the `XMLHttpRequest` object but also the DOM methodology differs greatly from browser to browser (especially with older versions). This requires extensive cross platform and version testing.

[69] http://code.google.com/intl/de-DE/web/ajaxcrawling/docs/getting-started.html – requested in March 2011

- Security Concerns:

  Due to their close relation to traditional HTML applications AJAX applications are vulnerable to cross-site scripting[70] and cross-site request forgery[71]. Although “XMLHttp” requests have a better security model than ordinary HTTP requests, the returned JavaScript is still a valuable target for malicious code.

- Browser History and Bookmarks:

  When working with multi-step processes on a web page, e.g. ordering something online, people got used to click the “back” button to jump to a previous step (to undo an action) as each step is often comprised by a separate document identified by a URL. Parts of an AJAX application are represented by the same single URL. Regardless of which functionality the user selects or how he/she changes the application state, the location is always the same. That is why the “Back” button cannot be properly used to reload previous content from the same application. Also bookmarks cannot be created as they require mapping content to a certain URI. To enable the necessary deep linking, hash fragments (#) may be used (see section “SEO”).

- Animation and multimedia support:

  Animation support highly depends on the framework employed. Complex, path-based animations like in Flash cannot be achieved. AJAX provides no native support for multimedia playback; for this purpose it relies on a separate plug-in.

7.2 Evaluation of Flex

According to Forrester Research (cf. [Hamm10]) Adobe Flex is the most frequently used plug-in related RIA development platform. The ubiquitous Flash Player and Flex’s integrated support for visual media design products of the Adobe family are responsible for the proliferation of Flex.

[70] http://en.wikipedia.org/wiki/Cross-site_scripting – requested in November 2010
[71] http://en.wikipedia.org/wiki/Cross-site_request_forgery – requested in November 2010

The security model of AIR, the necessary initial investment and the advent of HTML 5, however, may militate against choosing Flex.

Strengths

- Widespread adoption of Flash:

  Enterprises selecting Flex name the penetration of the Flash Player their key decision driver. When Flash is already installed, application deployment requires no additional changes to the user's desktop. Unlike Ajax, Flash doesn't require browser-specific code or enablement of JavaScript in the user's browser. (cf. [Hamm06]).

- Flash Builder:

  Although the Flash Builder comes with costs, it is a sophisticated IDE (identified as the most sophisticated during the project work) for developing RIAs. Its integrated support for products like Photoshop, Fireworks, Catalyst and Flash eases the collaboration of programmers and designers. A WYSIWYG-editor and wizards for data service integration encapsulate plenty of code and allow building a RIA quickly.

- Suitable for large-scale business applications:

  Various architectural frameworks (e.g. Cairngorm[72]) exist for Flex which favor processing large amounts of data and impose structure. Adobe LiveCycle[73] modules or BlazeDS[74] can be incorporated for business process automation and efficient data exchange with the back end.

- Offline and out-of-browser capabilities:

  Deploy Flex applications with AIR to provide RIA functionality beyond the browser borders. AIR applications can operate both in connected and disconnected mode and interrelate server data with the local file system.

[72] http://sourceforge.net/adobe/cairngorm/home/ – requested in February 2011
[73] http://www.adobe.com/products/livecycle/ – requested in December 2010
[74] http://opensource.adobe.com/wiki/display/blazeds/BlazeDS – requested in December 2010

Weaknesses

- ActionScript deficits:

  ActionScript 3 is strongly typed and very similar to Java but it misses abstract classes, private constructors, enums, generics and method overloading. The ActionScript compiler is slower than the C# compiler and less effective at diagnosing issues (e.g. detecting memory leaks).

- HTML 5 on the horizon:

  With the introduction of HTML 5 the audience reached by the Flash Player may decrease as HTML 5 acquires some key features of Flash (cf. chapter 8). People will most likely embrace an open standard which provides the same functionality as a proprietary plug-in.

- AIR security concerns:

  Web applications basically run in the sandbox of the browser to prevent malicious scripts from harming the local system. AIR applications must be signed but do not require a certificate from a recognized authority. Although Adobe may wants to free developers from going through the process of obtaining a legitimate certificate, distributing privileged applications without the need to rigorously authenticate them is a major security risk (cf. [Hals08]). Third-party software is always a potential host of malicious scripts which can cause significant harm if granted access to the local file system.

7.3 Evaluation of Silverlight

Silverlight developers benefit through decreased development time and decreased maintenance costs. If you primary have expertise in .NET, Silverlight is a good option for designing a Web application front end for an n-tier application system. Developers who are not familiar with .NET and respective languages are confronted with a steep learning curve. Developing on a platform different from Windows is impractical (see “Weaknesses” below).

Strengths

- .NET platform eco-system:

  Silverlight applications have access to a vast number of .NET libraries. Since virtually any CLI language can be used to write the application logic, Silverlight may require you to learn a single new language, XAML, which is self-explanatory to a large extend due to its XML nature. The .NET developer community is very large and offers valuable support and tutorials making Silverlight suitable for beginners.

- Imposes clear view/logic separation:

  In Silverlight prohibits writing application logic in the same file where the UI markup resides. The static appearance (GUI) is entirely written in `.xaml` files while the so-called code behind (the logic) has to be placed in corresponding `.xaml.cs` files. The benefit is that designers and programmers can work on the same application without having to deal with code from the domain of each other. Silverlight automatically arranges the linking of view and logic files.

- HTML integration:

  Any XAML element of a Silverlight application can be manipulated by the surrounding JavaScript code while the Silverlight code is privileged to interact with JavaScript without artificial boundaries.

- Computational performance:

  Due to just-in-time compilation and the CLI, which executes Silverlight as native code, Silverlight runs generally faster than Flex code. In contrast to Flex Silverlight supports real multi-threading; this can be very useful for processing computationally intensive code while it is vital to not block the GUI.

- Out-of-browser (OOB) capabilities:

  Silverlight OOB cannot challenge Adobe AIR but can be useful when an application should be available offline. The isolated storage at the user directory of the local machine the OOB model grants can be used for simple reading and writing operations on that part of the file system.

Weaknesses

- Plug-in distribution and acceptance:

  The plug-in is not part of any Windows OS distribution. It has to be downloaded separately or via Windows Updates. There is still a very significant percentage of Web users that does not have Silverlight installed (cf. figure 7 in chapter 2.4.1). A Web site would not be able to reach that audience with content relying on Silverlight (unless the content encourages them to install it).

- Development platform and platform support:

  Silverlight runs in every major browser as long as you use a Windows or Mac operating system. Although a Linux version named Moonlight[75] exists, it is excluded by Linux certain distributions like Fedora claiming patent and redistribution concerns[76]. Developing Silverlight applications on the Windows platform cannot be avoided since Visual Studio and Expression (except an outdated light version) only run on top of Windows.

7.4 Evaluation of JavaFX

JavaFX is mostly attractive to Java enthusiasts and people who are familiar with Java GUI building but desire a simpler alternative to complex Swing. The declarative syntax language makes designing a GUI easy and intuitive, saves a lot of code required to connect controls and provides a more attractive look and feel than

[75] http://www.go-mono.com/moonlight/ – requested in January 2011
[76] http://fedoraproject.org/wiki/ForbiddenItems#Moonlight – requested in March 2011

Swing. JavaFX excels when it comes to data binding and enriches the Java world with simple ways for connecting to Web services.

The major drawback of JavaFX is its uncertain future. Still an emerging technology, JavaFX will be revamped in 2011; JavaFX Script will be discontinued while parts of the JavaFX API are going to be integrated into Java. It remains to be seen how the revamped JavaFX performs.

Strengths

- Broad reach:

  Since the Java Virtual Machine is available virtually on every client machine, you do not have to find a way to encourage users to download the runtime environment your RIA requires.

- Java eco-system:

  JavaFX has holistic access to the Java API. It is even possible to create and employ custom Swing components in JavaFX. The fact that Java classes can be directly addressed by JavaFX has a huge impact on back end integration: connect a JavaFX GUI seamlessly with the entire business logic of a Java back end. This allows passing strongly typed data on service calls.

- Declarative syntax:

  Although the syntax for designing GUIs is not XML-based, it still has very much in common with MXML and XAML. A GUI in JavaFX requires less code, is more transparent and is more intuitively to design than a Swing/AWT GUI written in pure Java.

- Free of charge:

  Simply, but strong influencing factor: the JavaFX ecosystem is free of cost.

Weaknesses

- Tool support:

  JavaFX faces strong mature competition supported by specialized design tools. Flex and Silverlight are both powered by tools like Adobe Flash Catalyst and Microsoft Expression Blend which allow division of labor and simple integration of results as all tools related to a technology share the same project format. Although NeatBeans and Eclipse feature plug-ins granting syntax highlighting, refactoring and auto completion, JavaFX specific debugging and error identification requires improvements and still more bug fixes; tool support in case of JavaFX is generally restrained.

- No clear direction for JavaFX Mobile:

  JavaFX Mobile aims to replace Java ME and to provide a domain specific language (DSL) for designing GUIs of “write once, run everywhere” applications. The majority of mobile device vendors claims that JavaFX Mobile is not featuring characteristic device decided UI controls while JavaFX Mobile’s performance is not sufficient for running common App Store or Android Market applications and so only two Windows Mobile 6 devices (LG Incite and HTC Touch Diamond) feature JavaFX Mobile at the moment. Oracle is planning to introduce a JavaFX MSA (Mobile Service Architecture) player to enable JavaFX Mobile on more devices but for now placed JavaFX Mobile on hold.

- Graphics and multimedia:

  Rendering advanced 2D and 3D graphics inside the Java VM consumes much more CPU power than with Adobe Flash[77]. The JavaFX API provides a native media player for video playback which - for now - is not able to interact with the graphics kernel for acceleration. As a consequence, high CPU utilization is reached when playing standard definition (SD) videos, while 720p+ high definition (HD) playback cannot be achieved with acceptable frame rates.

[77] cf. http://geeknizer.com/why-choose-javafx-how-to-code-benchmark-graphics-cpu-memory/ – requested in April 2011

7.5 Recommendations

Each of these technologies has the same goal: bridging the gap between the Web and the desktop conglomerate through interconnectivity combined with intuitive GUIs offering immediate and accurate feedback. But their principle is not just to deliver fancy, rich GUIs. A RIA built with one of these technologies is supposed to aggregate and visualize complex data from different sources for intuitive manipulation by the client and to provide a specialized functional asset that engages the client and supports him/her on complex processes.

No one technology is inherently better than another. It is crucial to identify the needs of the clients regarding performance, interactivity and security. Each framework has to be measured in terms of maintainability and scalability and how well it integrates with legacy systems. Introducing a new technology can have a huge impact on productivity especially when in-house expertise does not match the required skills.

The following guidelines categorize AJAX, Flex and Silverlight according to use cases (cf. [Hamm06] and [Guir09])[78].

AJAX for incremental, tactical improvements, SEO-friendliness and customizable footprint

Select AJAX when release cycles are short and updates are released on a frequent basis. Knowledge about AJAX should easily be acquired by common Web developers. AJAX favors gradual refactoring of existing applications. AJAX frameworks and libraries can easily be changed and customized to meet specific application footprints.

[78] JavaFX is excluded as its platform and API are currently suitable for experimenting with small RIAs but not stable enough for sophisticated enterprise development.
See http://www.whippetcode.net/news/java-fx-enterprise-ready-/ – requested in February 2011 – for more information on it.

Flex for large-scale, more comprehensive user productivity applications

When building applications from scratch, Flex is a smart choice because Flex has integrated support for visual design tools like Catalyst and Photoshop, allows building custom component libraries and features mature architectural frameworks that allow imposing a high degree of individual structure on projects. Combine visual attractive and engaging Flash content with Flex controls and forms connected to services and data stores.

Adobe is a safe choice for those who place a high value on reliable support and product continuity, concerns that are vital when making strategic technology investments [Hamm06].

Silverlight for integration with .NET environment

Microsoft .NET is the most frequently used platform for deploying enterprise RIAs (cf. [Hamm10]). Therefore, Silverlight front ends will appeal most to enterprises working in .NET.

8 HTML 5 Outlook

HTML 5 is the successor of HTML 4 and XHTML 1.x. Its primary goal is to enrich the HTML conglomerate with new syntax and semantic elements for better structuring and to ensure that all major browsers offer the same functionalities and capabilities. HTML as we know could not be considered state-of-the-art. Vendors are forced to test malformed code across various browsers and then need to reverse-engineer their error-handling scheme. Semantic is also not a strength of HTML 4, since today's Web pages are filled with countless `<div>` tags which neither provide much information about the structure of the document nor are easy to read for both humans and computerized systems (search engines). Proprietary extensions (plug-ins) were chosen to fill technology gaps in the sense of interactivity and multimedia. Now with built-in tags for audio and video replay HTML 5 will have a huge impact on Adobe Flash and its unique selling proposition – the broad availability and multimedia media features.

The driving forces behind HTML 5 are the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG). All major browser vendors except Microsoft are members of the WHATWG.

According to Ian Hickson, editor of the HTML 5 specification and Google employee (his co-editor David Hyatt works for Apple), a major goal of HTML 5 is to move the Web away from proprietary technologies. He says: *"They're single-vendor solutions [and] they don't really fit well into the Web platform […] With an open platform, there's no such risk, since we have true competition, many vendors, and an open standard that anyone can implement […] It would be a terrible step backward if humanity's major development platform [the Web] was controlled by a single vendor the way that previous platforms such as Windows have been."*[79]

HTML 5 is expected to be especially embraced on the mobile sector as the only sophisticated cross platform solution since proprietary rich media technologies could not penetrate the mobile ecosystem not at least due to politically motivated barriers. For instance, HTML 5 contributes very well to Apple's decision to abandon

Flash from its smart phones. Apple claims Flash is supposed to run slowly and highly energy consuming on mobile devices. The more meaningful reason behind this strategy is that Flash would interfere Apple's business model concerning its electronic market places[80].

However, this chapter is not dedicated analyzing strategies of software vendors but analyzing new features of HTML 5 and especially the question if HTML 5 poses a potential threat to state-of-the-art RIA development frameworks. *Please note that HTML 5 is still in working draft stage. All features presented and assumptions made in this chapter are based on the HTML 5 roadmap, early hands-on experiences with demos and forecasts made by Forrester Research and are subject to change.*

The (Expected) Features of HTML 5

As of first quarter 2011, major browsers are slowly adopting some of HTML 5's new features. The following list points out the main characteristics of HTML 5[81].

- Enhanced Multimedia Support:

  The Flash Player produced relief for the Web plagued with incompatible technologies for multimedia playback. `<video>` and `<audio>` tags are designed for device and plug-in independent multimedia consumption and try to make the Flash Player obsolete. The video portal and Google subsidiary YouTube is already running a trial with HTML 5. A common audio and video format is not likely to be part of the specification as browser vendors try to push their own formats. A `<figure>` tag displays images but also code snippets along with a legend.

[79] http://www.infoworld.com/d/application-development/html5-could-it-kill-flash-and-silverlight-291?page=0,1 – requested in March 2011
[80] http://tech.fortune.cnn.com/2010/01/29/behind-the-adobe-apple-cold-war/ – requested in March 2011
[81] cf. http://www.drweb.de/magazin/html5-ueberblick/ (German) and http://ajdotnet.wordpress.com/2011/03/10/html5-part-ii-new-standard-for-web-applications/ – both requested in March 2011
Additionally you may want to pay a visit to http://www.html5rocks.com (as of March 2011) for a fancy but also quite good interactive introduction to HTML 5.

- Drawing:

  A `<canvas>` tag for 2D (and 3D) drawing and JavaScript enhanced animation should address use cases of the Flash domain. A `<svg>` tag supports inline Scalable Vector Graphics (SVG).

- Semantic Tags:

  Tags like `<header>`, `<footer>` and `<nav>` and descriptive link relations shall provide a fine-grained identification of document parts, replace countless `<div>` tags and support easier navigation. A `<time>` tag provides machine readable information on vague date specifications.

- CSS3[82]:

  Define interactive behavior with transitions, transformations and animations; graphical enhancements like gradients and shadows.

- Specialized Forms with Client-Side Validation:

  New types for the `<input>` element, e.g. `date` or `email`, provide basic validation. Define valid regular expressions and mark fields as required or subject to auto completion.

- Improved Web Storage:

  Use `setItem` and `getItem` on `window.localStorage` to store data locally in an isolated storage. An application cache stores the status of a web page for offline use and can be set to reload a page just in case of an update from the Web. HTML 5 also comes with a Web SQL Database API.

- Geolocation:

  Builds on Google Gears and should provide a standardized interface to retrieve geographical location information.

[82] See http://wi.wu-wien.ac.at:8002/rgf/diplomarbeiten/Seminararbeiten/2010/201007_Piffer/201007_Piffer_CSS.pdf for a brief introduction to CSS3.

- Web Workers:

  Implement long-running, complex scripts to run on a background thread without interfering user interaction on the main thread. Use `onmessage()` method for communication between main thread and workers.

- WebSocket:

  Establishes full-duplex communication for pushing updates to the server and pulling updates from the server without (significant) delays; plain old AJAX is not designed for this. This concept known as “long polling” or “long live request” (umbrella term “Comet”) would require continuously initiating and sending new `XMLHttpRequests`, thus producing a large amount of overhead.

- Native Drag & Drop and FileSystem Access:

  Invoke the `datatransfer` object on a JavaScript event to get access to functionality for handling drag & drop gestures inside the browser but also from the browser to the desktop and vice versa. Use the FileSystem API to work with a sandboxed section of the local file system.

Under Construction

As of first quarter 2011 HTML 5 is available as working draft. A candidate recommendation stage will not be reached before 2012 while a stable, ratified W3C standard is expected in 2014[83]. In the meantime HTML 5 is challenged by inconsistent behavior across browsers and platforms. For instance, the way how HTML 5 renders complex objects leaves room for interpretation and leads to varying results among browsers. Since HTML 5 is still in development, it will of course take some time before standard control libraries appear and major IDEs support HTML 5 with WYSIWYG editors.

[83] cf. http://wiki.whatwg.org/wiki/FAQ#When_will_HTML5_be_finished.3F – requested in March 2011

RIA Frameworks Persist

The prevalent opinion[84] is that HTML 5 will challenge today's RIA technologies but will not be able to substitute them in the near future. Although HTML 5 is supposed to excel in terms of broad reach, system requirements, basic controls and mobile support, RIA frameworks are just more mature; see figure 37 for an early comparison by Forrester Research supposed to give the reader an idea on how HTML 5 would compete with RIA technologies if it were released today.

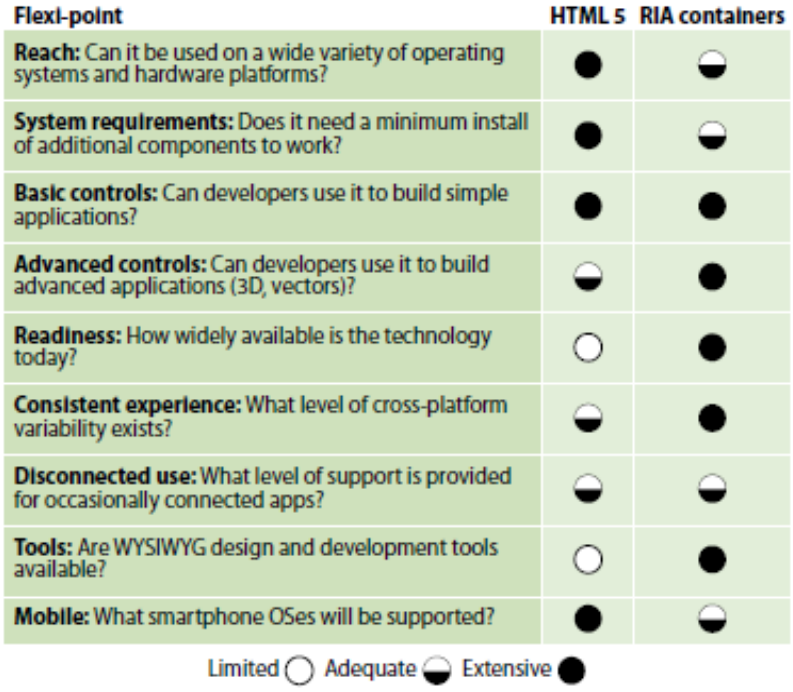


Figure 37: HTML 5 vs. RIA Frameworks [Hamm10]

In order to be attractive to enterprise developers, HTML 5 has to be deployable at low cost with a support level and functional asset comparable to frameworks like Flex or Silverlight. In addition, it needs to seamlessly integrate with existing back ends and legacy systems. HTML 5 constitutes an evolution not a revolution. It neither provides a new programming model nor changes server side processing. It will still rely on AJAX's `XMLHttpRequest` for efficient asynchronous HTTP communication with a server. WebSockets require a WebSockets server and a secured socket

[84] See http://active.tutsplus.com/articles/roundups/html5-and-flash-17-industry-experts-have-their-say/ – requested in May 2011 – for a representative interview with 18 experts on the field of RIA development and Web design.

connection. Unless requested data is changing constantly, WebSockets would generate much more overhead than AJAX and consequently would carry asynchronous requests to excess. HTML 5 will also resort to AJAX widget toolkits for controls not built-in. The expected functional asset of HTML 5 is still missing vital features sophisticated business applications require[85]:

- Processing large amounts of data; data input with complex validations, visual controls for smart data aggregation
- Immediate feedback (not possible with post backs), dynamic UIs with forms built dynamically or changing depending on user input
- Real offline and out of browser capability
- Stateful programming model, MVC patterns with data binding mechanisms

The big advantage of proprietary RIA solutions is that they advance at a much faster pace than HTML and will always be ahead on the curve when compared to open standards since innovation is not restrained by standards bodies and opposing corporations.

Pressure Makes Diamonds

HTML 5 will definitely weaken the position of Adobe Flash. The Flex framework profited from the fact that the necessary runtime (Flash Player) was installed on virtually every system. HTML 5 still has a long way to go before browser consistently support tags and UI controls while RIA frameworks evolve further. The bottom line is that HTML 5 and today's RIA frameworks are likely to coexist, serving different purposes: HTML 5 will accelerate the "Semantic Web" movement and will be used for multimedia playback especially on mobile devices; RIA frameworks will still be employed as front end of enterprise Web applications.

[85] cf. http://ajdotnet.wordpress.com/2011/03/12/html5-part-iii-the-limits/ – requested in March 2011

9 Round-up and Outlook

This thesis aimed to introduce the reader to Rich Internet Applications which basically replace traditional thin web application clients and try to fill the gap between the web and the desktop conglomerate. It outlined the advantages of a RIA and general principles a web application should follow to generate maximum user satisfaction. The essential artifacts of every RIA are rich GUI components and mechanism for asynchronously communicating with a server, thus, retrieving (further) application data without artificially interrupting the user's workflow. RIA functionality goes beyond capabilities of traditional web applications. A RIA actively takes over rendering and even logic processing duties from the server.

RIA development frameworks put in great effort to simplify tasks like the process of designing a GUI with a declarative language or consuming remote web services. The trailblazer AJAX is still the most popular RIA framework. Combining web standards with its AJAX engine and open source widget libraries, it provides a convenient experience and reaches the widest audience.

However, AJAX faces strong competition from Flex and Silverlight. These technologies employ their own runtime and are part of the Flash and .NET development platform respectively. Flex and Silverlight applications tend to be more suitable for presenting rich media and for integration with enterprise systems. Additionally both come with great IDE support. While AJAX applications are restricted to the browser, Flex and Silverlight advance to the desktop sphere as they provide continuous application support both in connected and disconnected mode and enable interaction with the local system. Java introduced JavaFX to catch up with mature RIA technologies and to revamp Java Applets. Although JavaFX has great potential, for instance it features various mechanisms for fine-grained data binding; it could not take off so far. Moving JavaFX to the Java API will definitely help JavaFX to appeal well to the Java community.

HTML 5 is going to have an impact on the RIA conglomerate as it addresses key features of the domain of plug-in based technologies. But HTML 5 is still on the advent. A significant comparison to Flash and Silverlight first requires HTML 5 to be released as stable W3C standard.

10 References

[AIRComp] Browser vs. application

http://www.adobe.com/products/air/comparison/

Requested in November 2010

[Brown10] Brown P.: Silverlight 4 in Action. Stamford 2010: Manning Publications Co.

[BuKo09] Busch M., Koch N.: Rich Internet Applications – State-of-the-Art. Ludwig-Maximilians-Universität München, December 2009

[CaGh10] Cameron R., Ghosh J.: Silverlight Recipes: A Problem-Solution Approach, Second Edition. New York 2010: Springer Science+Business Media, LLC

[CCB09] Clarke J., Connors J., Bruno E.: JavaFX – Developing Rich Internet Applications. Addison-Wesly 2009

[DBBO07] Deb B., Bannur S. G., Bharti S.: RIA – Opportunities and Challenges for Enterprises. Infosys White Paper, January 2007

[DeveAJ] AJAX: Asynchronous Java + XML?

http://www.developer.com/design/article.php/3526681/AJAX-Asynchronous-Java--XML.html

Requested in November 2010

[DVPG05] Driver M., Valdes R., Phifer G.: Rich Internet Applications Are the Next Evolution of the Web. Gartner, May 2005

[EDWF10] Fain Y., Rasputnis V., Tartakovsky A.: Enterprise Development with Flex. 2010

[Garr05] Garrett J. J.: Ajax: A New Approach to Web Applications. February 2005

http://www.adaptivepath.com/ideas/essays/archives/000385.php

Requested in November 2010

[GhSc09] Ghoda A., Scanion J.: Accelerated Silverlight 3. New York 2009: Springer Verlag

[Giur09] Giurata P.: Rich Internet Application agnosticism - Flex vs. Silverlight vs. AJAX / HTML5. November 2009

http://www.catalystresources.com/saas-blog/rich_internet_application_agnosticism/

Requested in February 2011

[HaGo07] Hammond J., Goulde M.: Rich Internet Apps move beyond the browser. Forrester, June 2007

[Hals08] Halsstead B.: Why Criminal Hackers Will Love Adobe AIR

http://developers.curl.com/blogs/community_blog/2008/04/16/why-criminal-hackers-will-love-adobe-air

Requested in November 2010

[Hamm06] Hammond J. S.: Ajax Or Flex – How To Select RIA Technologies. Forrester Research

[Hamm10] Hammond J. S.: Does HTML 5 Herald The End Of RIA Plug-Ins? Forrester Research

http://www.adobe.com/content/dam/Adobe/en/enterprise/pdfs/html-5-ria-plug-ins.pdf

Requested in February 2011

[Hein08] Heinrichs M: Using JavaFX Objects in Java Code, March 2008

http://blog.netopyr.com/2008/03/21/using-javafx-objects-in-java-code/

Requested in February 2011

[Hold08] Holdener A. T.: Ajax – The Definitive Guide, O'Reilly. January, 2008

[InterAJ] Interakt Online

http://www.interaktonline.com/files/art/ajax/AJAX%20-%20Asynchronously%20Moving%20Forward.pdf

Requested in October 2010

[IsecAJ] Information Security Partners: Attacking AJAX Web Applications

https://www.isecpartners.com/files/iSEC-Attacking_AJAX_Applications.BH2006.pdf

Requested in November 2010

[JenkJR] Jenkov J.: Java Reflection Tutorial

http://tutorials.jenkov.com/java-reflection/index.html

Requested in February 2011

[KeCh09] Keefe M., Christiansen C. A.: Java and Flex Integration Bible. Wiley 2009

[Mull03] Mullet K.: The Essence of Effective Rich Internet Applications. Macromedia Whitepaper, November 2003

[LRWA08] Linnenfelser M., Rech J., Weber S.: An Overview of and Criteria for the Differentiation and Evaluation of RIA Architectures. March 2008

[Moock07] Moock C.: Essential ActionScript 3.0. O'Reilly 2007

[NoHe05] Noda T., Helwig S.: Technical Comparison and Case Studies of AJAX, Flash, and Java based RIA. Best Practice Reports, University of Wisconsin-Madison, November 2005

[OfJe02] Offutt J: Web Software Applications Quality Attributes. George Mason University, November 2002

[Treto08] Tretola R.: What is RIA? January 2008

http://insideria.com/2008/01/what-is-ria-1.html

Requested in October 2010.

[Tric08] Trice A.: Understanding The Architecture of a Rich Internet Application. February 2008

http://insideria.com/2008/02/understaning-the-architecture.html

Requested in October 2010.

[WGCI09] Weaver J., Gao W., Chin S., Iverson D.: Pro JavaFX Platform – Script, Desktop and Mobile RIA with Java Technology. New York 2009: Springer-Verlag

[Wick08] Wick B.: RIAs Integrated with Enterprise Systems Deliver Greater Value to Businesses. May 2008

http://www.information-management.com/infodirect/2008_72/10001286-1.html

Requested in October 2010

[Widj08] Widjaja S.: RIA Entwicklung mit Flex 3. Hanser March 2008

[WijnAJ] Wijngaarden T. van: Asynchronous JavaScript and XML

http://www.few.vu.nl/~eliens/projects/design-/multimedia/@archive/student/archive/lit-teunis.pdf

Requested in November 2010