Study and Development of Rich Internet Applications
Project Report

Pedro Melo Campos
MIEIC 2008
Study and Development of Rich Internet Applications

Pedro Melo Campos

Project Report
Master in Informatics and Computing Engineering
Supervisor: Luís Paulo Reis, PhD

2008, July
Study and Development of Rich Internet Applications

Pedro Melo Campos

Project Report
Master in Informatics and Computing

Approved in a public examination by the Jury:
Chair: António Fernando Castro Coelho, PhD

External Examiner: Nuno Magalhães Ribeiro, PhD
Internal Examiner: Luís Paulo Reis, PhD

2008, July
Abstract

Rich Internet Applications (RIA) follow recent software development and deployment paradigms that, as the name implies, have improved the quality of web-related applications. Technologies initially associated with web applications have matured in the last few years, to such extent that they are becoming the standard for interface development, even for desktop applications. As the gap between desktop and web application narrows, there is a deployment convergence that goes beyond these two worlds, beginning to include mobile applications.

This project objective includes the choice of a RIA technology, a description of how to apply Software Engineering practices to RIA development and the design and implementation of highly modular components and applications for a professional video software company, Media Objects and Gadgets, S.A. (MOG).

The choice of a particular RIA technology carries advantages and costs, which aren’t always obvious. To provide the most thorough analysis of the existing technologies, a research on the background concepts of RIA development, as well as its history and predictions on its future was conducted. The result of this study defines a large number of parameters on which to compare the technologies and offers a framework for choosing between three leading RIA technologies (AJAX, Silverlight and Flex) according to the needs of the project where the technology will be used.

The implementation of RIA has its own specificities, but many of the web or desktop software engineering practices can and should be included in a proper RIA development workflow. Identifying these important best practices is very important in adapting to the RIA environment, and this project aims to establish a strong basis for a good RIA development.

Several professional media applications were developed at MOG using Adobe Flex to develop the front-end, often connected to existing and in development back-end technologies via web services. These applications are deployed both on the desktop and the web, and use XML extensively. Three of these applications - Tobogган Monitor, Tobogган XDCAM Proxy Merger and Tobogган EDLConverter - are part of the Tobogган series of products, controlling and monitoring the automation of digital media related tasks. XMLEasyConfig is an application that dynamically creates an interface for editing XML based on the XML Schema that describes it. Finally, MXF SpeedRail Metadata Editor is an ingesting and logging component of a media asset manager. With the exception of XMLEasyConfig, which is a work in progress internal tool, these applications were deployed with success, passing the quality tests and requirements of important clients, with Tobogган products being used in the Beijing Olympics.

Summarizing, this project approaches RIA from many different angles, from theoretical to practical and from high level thinking to implementation level patterns, providing a reference and framework for RIA development.
Resumo

As Rich Internet Applications seguem recentes paradigmas de desenvolvimento e distribuição que trazem melhorias às aplicações ligadas à web. Tecnologias anteriormente associadas às aplicações web amadureceram de tal forma nos últimos anos que estão a tornar-se o padrão para desenvolvimento de interfaces, mesmo para aplicações de computador. À medida que a distância entre o ambiente desktop e a web se encorta, nota-se uma convergência na distribuição de aplicações que começa a incluir os dispositivos móveis.

Este projecto inclui a escolha de uma tecnologia RIA, uma descrição de como aplicar práticas de engenharia de software ao desenvolvimento de RIA e o planeamento e implementação de componentes e aplicações modulares para uma companhia de software de vídeo professional, a Media, Objects and Gadgets, S.A. (MOG).

A escolha de uma tecnologia RIA em particular acarreta benefícios e custos que nem sempre são óbvios. Para que a análise das tecnologias existente seja o mais aprofundada possível, fez-se uma investigação dos conceitos por trás do desenvolvimento de RIA, da sua história e do possível futuro. O resultado deste estudo define os parâmetros em que as tecnologias são comparadas, estabelecendo uma plataforma para a escolha entre três tecnologias RIA líderes (AJAX, Silverlight e Flex) tendo em conta as necessidades do projecto em que a tecnologia é utilizada.

A implementação de RIA tem as suas especificidades, mas muitas das práticas da engenharia de software podem e devem ser incluídas num ambiente de desenvolvimento RIA cuidado. Identificar estas práticas é um passo importante na adaptação à envolvente das RIA e este projecto procura estabelecer uma base sólida para uma boa programação de RIA.

Foram desenvolvidas várias aplicações na área dos media profissional na MOG, utilizando o Adobe Flex para implementar o front-end, frequentemente ligado por serviços web a tecnologias back-end já existentes ou em desenvolvimento. Três dessas aplicações - Toboggan Monitor, Toboggan XDCAM Proxy Merger e EDLConverter - controlam e monitorizam a automação de tarefas de media digital. XMLEasyConfig é uma aplicação que cria dinamicamente uma interface para edição de XML baseada no XML Schema que define o seu tipo. Por fim, MXFSpeedRail Metadata Editor é um componente para ingest & logging de um gestor de conteúdos multimédia. A exceção do XMLEasyConfig, que é um projecto interno ainda em desenvolvimento, as aplicações foram distribuídas com sucesso, tendo passado nos testes de qualidade e requisitos de

III
importantes clientes, com os produtos Toboggan a serem usados nos Jogos Olympicos de Pequim.

Resumindo, este projecto debruça-se sobre a temática RIA de vários ângulos, do teórico ao prático e do pensamento de alto nível aos padrões de implementação, oferecendo uma referência e base ao desenvolvimento de RIA.
Acknowledgements

I would like to thank my supervisor at FEUP, Luís Paulo Reis, and my supervisor at MOG, Vitor Teixeira, for their advices on writing a thesis.

This work would not have been possible neither without my family, friends and loved ones, nor without LEIC, MIEIC, MOG and all my colleagues. Thanks for shaping this brain I love to inhabit.
# Contents

Abstract .............................................................................................................. I
Resumo ............................................................................................................... III
Acknowledgements .......................................................................................... V
List of Figures ................................................................................................... IX
List of Tables ..................................................................................................... XI
Acronyms .......................................................................................................... XIII

2. Introduction .................................................................................................. 1
   2.1. Scope ...................................................................................................... 1
   2.2. Motivation .............................................................................................. 1
   2.3. Objectives .............................................................................................. 2
   2.4. Structure ............................................................................................... 2

3. Introducing Rich Internet Applications ....................................................... 3
   3.1. Desktop applications, web applications and RIA .................................. 3
   3.2. Examples of RIA ................................................................................... 4
   3.3. Conclusion ............................................................................................. 6

4. The RIA Technologies ................................................................................ 7
   4.1. Model, Layout, Styles and Actions using Open Standards .................. 7
   4.2. AJAX ..................................................................................................... 8
   4.3. Traditional Three-Tier Architecture ...................................................... 9
   4.4. Service Oriented Architectures ............................................................. 10
   4.5. Event-Driven Architectures .................................................................. 10
   4.6. Technological Requirements for Online Applications ....................... 11
   4.7. Post-AJAX Technologies ...................................................................... 12
   4.8. Advantages of XML-based Languages ............................................... 13

VII
List of Figures

Figure 2.1: Example of a RIA - Songbird .................................................. 5
Figure 2.2: Example of a RIA - Picnik .................................................. 5
Figure 3.1: HTML and CSS Rendered .................................................. 8
Figure 3.2: Three-tier Architecture ................................................... 9
Figure 3.3: Event handling sequence example ...................................... 10
Figure 3.4: Data Binding Example in Flex .......................................... 12
Figure 3.5: XML-Based Flex MXML example ....................................... 13
Figure 5.1: Declarative XML instantiation and data binding .................... 22
Figure 5.2: The same object created in MXML and in ActionScript .......... 22
Figure 5.3: Suggested code structure ................................................ 23
Figure 5.4: Example of the mediator pattern applied in Flex ................. 24
Figure 5.5: Model, View, Controller and Services ............................... 25
Figure 6.1: Two different XMLs rendered in XML Navigator .................. 34
Figure 6.2: Connection Status component .......................................... 35
Figure 6.3: Observer component and example of its use ........................ 35
Figure 6.4: Toboggan backend Architecture ....................................... 38
Figure 6.5: Toboggan Monitor use cases ............................................ 39
Figure 6.6: Monitor Configuration XML example .................................. 41
Figure 6.7: Toboggan Monitor Implementation Patterns ......................... 42
Figure 6.8: Toboggan Monitor ......................................................... 43
Figure 6.9: Toboggan Processing Chains ............................................ 43
Figure 6.10: XDCAM Proxy Merge .................................................. 44
Figure 6.11: XDCAM Proxy Merge .................................................. 45
Figure 6.12: XDCAM Controller configuration ................................................. 45
Figure 6.13: EDL Converter sequence diagram ................................................. 46
Figure 6.14: EDL Converter old UI ................................................................. 47
Figure 6.15: EDL Converter new UI ................................................................. 47
Figure 6.16: Browsing in EDL Converter, using a submit popup component ....... 48
Figure 6.17: Possible Future of Toboggan ......................................................... 50
Figure 6.18: XMLEasyConfig sequence diagram .............................................. 51
Figure 6.19: XMLEasyConfig Layout Specification .......................................... 52
Figure 6.20: XMLEasyConfig components diagram ........................................ 52
Figure 6.21: XMLEasyConfig Prototype and testing environment .................... 54
Figure 6.22: MFX SpeedRail Metadata Editor use cases .................................. 55
Figure 6.23: MXF SpeedRail Metadata Editor view/edit metadata use cases ........ 56
Figure 6.24: MXF SpeedRail Metadata Editor with several productions ............ 57
Figure 6.25: MXF SpeedRail Metadata Editor integrated with MOG player ....... 58
Figure A.0.1: Xerox 8010 Star UI ................................................................... i
Figure A.0.2: First WWW browser ................................................................ ii
Figure A.0.3: Browser usage during the browser wars ................................... iii
Figure A.0.4: Mac OS-X UI .......................................................................... iv
Figure A.0.5: Web 2.0 Timeline and Attributes ............................................. v
List of Tables

Table 4.1: RIA Technologies Reach Comparison.................................................. 16
Table 4.2: RIA Trust Comparison................................................................. 17
Table 4.3: RIA Costs Comparison................................................................. 17
Table 4.4: RIA Development Ease Comparison............................................. 18
Table 4.5: RIA UX Comparison ................................................................. 19
Table 4.6: RIA Comparison Summary......................................................... 20
Acronyms

AIR     Adobe Integrated Runtime
AJAX    Asynchronous JavaScript and XML
API     Application Programming Interface
AS      ActionScript
AVM     ActionScript Virtual Machine
CSS     Cascading Style Sheets
ECMA    European Computer Manufacturers Association
EDA     Event-Drive Architecture
EDL     Edit Decision List
ES      ECMAScript
GUI     Graphical User Interface
GWT     Google Web Toolkit
HTML    Hypertext Markup Language
IDE     Integrated Development Environment
IE      Microsoft's Internet Explorer
JavaEE/JEE Java Enterprise Edition, formerly J2EE
JIT     Just In Time
JS      ECMAScript, JavaScript or JScript
JRE     Java Runtime Environment
JSP     Java Server Pages
JVM     Java Virtual Machine
MID     Mobile Internet Devices
MVC     Model View Controller
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MXF</td>
<td>Material Exchange Format</td>
</tr>
<tr>
<td>NAB</td>
<td>National Association of Broadcasters</td>
</tr>
<tr>
<td>MOG</td>
<td>Media Objects and Gadgets, S.A.</td>
</tr>
<tr>
<td>OOD</td>
<td>Object Oriented Design</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PHP</td>
<td>PHP: Hypertext Preprocessor</td>
</tr>
<tr>
<td>RIA</td>
<td>Rich Internet Application</td>
</tr>
<tr>
<td>SDI</td>
<td>Serial Digital Interface</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WPF</td>
<td>Windows Presentation Foundation</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WYSIWYG</td>
<td>What You See Is What You Get</td>
</tr>
<tr>
<td>XAML</td>
<td>Extensible Application Markup Language</td>
</tr>
<tr>
<td>XBL</td>
<td>XML Binding Language</td>
</tr>
<tr>
<td>XHTML</td>
<td>Extensible Hypertext Markup Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XPath</td>
<td>XML Path Language</td>
</tr>
<tr>
<td>XQuery</td>
<td>XML Query</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensive Stylesheet Language Transformations</td>
</tr>
<tr>
<td>ZUI</td>
<td>Zoomable User Interfaces</td>
</tr>
</tbody>
</table>
1. Introduction

1.1. Scope

This project was developed at Media Objects and Gadgets, S.A. [MOG08] (MOG), as the final project of the Integrated Master in Informatics and Computing Engineering of the Faculty of Engineering, University of Porto. MOG develops professional media software, mainly related to the MXF standard, a container format for professional digital video and audio. Its products include MFX development tools, workflow automation, an ingest station and media asset manager and different solutions for conversion, annotation and distribution of media assets, working with many of the world leaders in the professional video production.

The focus of this project is on Rich Internet Applications (RIA). RIA combine functionalities from desktop applications and web applications, providing superior graphical user interfaces and interface design patterns, largely processed on the client side. They can be deployed on the web or on the desktop and usually take advantage of the internet or service oriented architectures.

1.2. Motivation

I chose this subject because of my belief that considering the web as the next platform for software delivery, whether or not an accurate prediction, potentiates the most interesting considerations regarding application scopes, business models, device independence and ubiquity and cloud computing, stimulating research in several areas of informatics. The innovation is only tamed by the infant state of the languages and technologies when compared to desktop development. The differences are narrowing though, and RIA technologies seem a very important tool for an informatics engineering student to have. I believe that I have filled a gap in this area.
1.3. Objectives

The first objective was to provide a reasonably study about the RIA environment so that some inferences could be made about the future and a decision could be made about which of the available technologies to choose. This was an important step in MOG Solutions’ technological strategy. After choosing the technology, it was important to get to know it quite well and understand the best practices, both architectural and procedural. The results of these considerations define internal standards and guidelines for future development with the technology, which is expected to be extensively used. Finally, a set of applications was developed for several MOG products, not as a proof of concept but in real life applications and for some important clients.

This document assumes that the reader has minor knowledge on the subject, as I was when I started it, but will go deep into considerations such as comparison of the leading technologies, predictions on the future of RIA, best practices and proposed workflow. Rather than just providing the details on the software developed, the objective will be to provide a good idea about most things related to RIA, and to Adobe Flex, the technology selected from the market analysis.

1.4. Structure

In accordance to the objectives, this document is has three main sections, the first being technology study and choice, followed by the Flex working environment and finally the developed applications sections.

The technology study and choice begins in chapter 2, defining the concept and characteristics of Rich Internet Applications. Chapter 3 discusses the technological side of RIA, the roles of different RIA languages, the shift in architectural paradigms currently happening and important events that have shaped RIA’s evolution. Chapter 4 summarizes the most important aspects that differentiate these technologies, as well as the reasons that led to the adoption of Flex.

Chapter 5 is the Flex working environment section which discusses common solutions for recurring implementation problems, applies known design patterns to Flex development and defines important steps in a Flex enterprise workflow.

Chapter 6 describes the development phase of the project, which consisted of several small to medium highly modular, dynamic and configurable applications for very different purposes and technological environments. Many reusable components were also created for future usage.

The project outcome will be presented in chapter 7 and will summarize all of the project aspects. Finally, the references and an annex describing the history of RIA can be found at the end of the document.
2. Introducing Rich Internet Applications

Over the last decade, the focus on web application interfaces has grown. The second wave of web businesses (web 2.0 [Shu08] [Tap08]) is due not only to new business models and user collaboration but also to the increased potential of the web as a complete delivery platform. Websites got enriched with functionality, making them behave like full-featured applications. Desktop Applications interact more and more with the Internet. Even the interface paradigms are converging. This chapter introduces the concept of Rich Internet Applications and provides some examples with most of its defining traits.

2.1. Desktop applications, web applications and RIA

The concept of Rich Internet Applications can be understood by comparing it to its roots, the desktop and web applications.

**Traditional desktop application** – A traditional desktop application is installed on the operating system and uses a hard drive for storage. It can serve many purposes, like information handling, media creation and visualization, education or recreation. Desktop applications can be completely independent of the internet; some though, may use the internet for sporadic functions, like downloading add-ons or loading data from a server. In this case, they are called internet-enabled applications or rich/fat/thick clients (e.g. a personal information manager, like MS Outlook [Mic08]).

**Traditional web application** – A traditional web application a website, loaded on a browser, personalized using logins. A constant connection is required, but no installation. Due to the web history, web applications navigation is traditionally page based.

**Rich Internet Application** – Both desktop and web applications have their limitations and strengths and the acknowledgement of such and the development of both server and client-side technology lead to the so-called Rich Internet Applications, the interface richness and the complexity of the desktop applications meets the connectivity and ubiquity of web apps. They can be deployed and updated seamlessly on the desktop or on the web, and make use of both. Web-deployed RIA aim lightness, so they can be loaded quickly and integrated with other sites in a mash-up (e.g. Google Maps [Goo08] or YouTube [You08] videos embedded everywhere).
Advantages of having an application running online:

- No distribution costs;
- Reach potentially over a billion users;
- Ubiquity – The user doesn’t need to use the application in his personal computer;
- No installation required.

RIA User Interfaces

The “rich” in RIA is associated not only with its functionality, but also with the interface design patterns. RIA can have a highly customized and carefully designed look, with complex visual components tailored to improve each specific task, far from the habitual web pages simple layouts. They have transitions, animations and embedded multimedia enhance the experience. There are no page refreshes, just a single dynamic and stateful page that includes all the parts of the application. The navigation and interaction include new patterns like drag-and-drop or mouse gestures, besides the traditional scrolling and clicking of buttons. [Mor08]

2.2. Examples of RIA

The best way to understand the power of RIA is to take a look at some examples. Widely known RIA include GoogleDocs [Goog08] or Gmail [Goog08]. In the next couple of pages, two commented examples that sum up most of the possibilities of RIA are presented.

Songbird

Songbird [Pio08] is a good example of RIA, but not an obvious one. Songbird is an open source, free media player. It is similar in many ways to the best known iTunes [App08], with additional features and freedom.

It is installed on the desktop, but uses the web for a lot of its functionality. It offers very complete programmer APIs for both desktop and web, meaning it can be easily extended and mashed-up (e.g. there are extensions that show Wikipedia’s description of the track/artist playing, or related Flickr [Yah08] pictures). It is integrated with eMusic [eMu08], an online music store (like iTunes’), and media search engines. A user can stream or download the media found through these searches, or use playlists of media in his desktop. As we can see, the online/offline distinction is blurred.

The interface is completely configurable and built over Mozilla’s XULRunner [Moz08], making it platform independent and standards-compliant. The default interface, also similar to iTunes, incorporates column browsing and auto-complete search, interface design patterns typical of the web, as it possible to see in figure 2.1.
Picnik

Another good example of RIA is Picnik [Pic08], an online photo editing service, developed with Adobe Flex. Picnik is integrated with most image hosting sites, like Flickr, allowing a complete workflow online, accessible everywhere.

An example of a typical traditional desktop application could be Photoshop, because of its complexity. Picnik offers most commonly used functionalities in Photoshop for photo editing, like effects, filters (see figure 2.2. for an example) and a lot of image composition tools.
2.3. Conclusion

Rich Internet Applications are applications that mix features and interface paradigms both from the desktop and the internet worlds. They can be deployed on the web (requiring no installation) or on the desktop. Usually they have nice-looking and responsive interfaces, use the interconnectivity of the web and run in a single page without reloads.

The next chapter will deal with the technology behind these applications.
3. The RIA Technologies

This chapter is about the technologies that allow the development of RIA. While web applications involve client and server technologies (and so do Rich Internet Applications deployed on the web), the main differences of RIA reside on the client side power. Starting with the basics, this chapter addresses the shift to the client-side and the motives behind it, explaining the new trends and trying to infer the future of RIA. It describes a multitude of different architectural considerations that move RIA, and should move a RIA developer.

3.1. Model, Layout, Styles and Actions using Open Standards

The current standard-based web development uses several different languages with different purposes. XML for the model and communication serialization, XHTML for the layout, CSS for styles and JavaScript for actions are the most important languages to know. Explaining these languages is outside the scope of this document, but I’ll provide an extremely simplified look at them, just the absolute minimum required to follow the rest of this chapter.

XHTML [Wor08] (Extensible Hypertext Markup Language) defines the layout of an interface at a given moment as well as some characteristics of the several components of it, e.g. what do buttons link to or what actions they trigger when pushed. The look of each of these components is defined by CSS [Wor08] (Cascading Style Sheets). With these languages it is possible to create a static page that a browser can read rendering the layout specified by the XHTML with the look defined by the CSS. Figure 3.1. provides an example of how CSS and XHTML syntax looks like and how this very simple code would be rendered by a browser.
XML [Wor082](Extensible Markup Language) is the standard specification for markup languages, and has many different implementations with different purposes (One example is XHTML). But by referring XML as the model and mean of communication, I mean XML as the wrapper of information. The web content to be displayed and used by an application is very often stored and supplied by the server in XML, e.g. text to be displayed. This content is loaded using a client-side scripting language, traditionally JavaScript [Fla08]. JavaScript can act upon XML, XHTML and CSS, changing them over time and in response to user interaction. This allows the creation of dynamic pages, where the look and data can change over time according to events and user actions.

This separation of concerns is a good practice, relating to the MVC design pattern (Actions being the controller and Layout and Styles the View). It makes changes easier and promotes reusability. An example of this reusability can be found in mashup sites that may incorporate parts of other sites and maintain their own look just by using their own CSS. Because of shortcomings in browser’s implementation of XHTML and CSS, and since JavaScript can act upon them, it is common to see the MVC pattern broken, with JavaScript (with its own browser incompatibilities and ugly hacks) creating the interface components and their presentation. The MVC pattern will be analysed in the Flex Working Environment chapter.

3.2. AJAX

A technology called remote scripting allows JavaScript to exchange information with the server and call server-side scripts. In the very late nineties, some intelligent minds started to notice the potential of this feature to enhance the dynamic nature of a page. Previously, while one could use JavaScript to change the model and view of the page without refreshing the browser, it wasn’t possible to get new models or views according to some user choice. Using remote scripting, it became possible to load chunks of data at whenever appropriate. This feature brought great freedom to the developer (not without some clever hard work) and started the possibility of
having a whole application running in a single page, loading the data from the server when appropriate, mutating programatically or even polling the server waiting for changes in a database to keep represented data up-to-date.

These ideas would later be renamed to AJAX [Hol08] – Asynchronous JavaScript and XML. “AJAX” became a widely used buzzword, and was responsible in great part for the pursuit of Rich Internet Applications. Actually, it became such a huge buzzword that it swallowed the whole standards-based client-side web development (AJAX is also used to refer the combination of XHTML, CSS, JavaScript and XML on a single page stateful application).

3.3. Traditional Three-Tier Architecture

Traditional web applications have a three-tier architecture: the database represents the data tier, the server represents the logic tier, and the page rendered on the browser the presentation tier. A diagram of this architecture and the usual requests and answers from each tier are shown in figure 3.2.

Database modelling can be made with different objectives in mind, the objective of avoiding replication, pursuing consistency and reducing the database size, using replication for protection against data loss or using views and data warehousing for special optimizations of particular uses of the database. Server languages running on the server will create a page on request (from a user’s browser), and process actions caused by the user interaction will request new pages, or update the data tier and the states of the user session. The page created will run on the browser and user interaction and input will be forwarded as requests to the server.

![Figure 3.2: Three-tier Architecture](image)

There are some similarities between the three-tier architecture and the Model View Controller pattern. The difference is that the view doesn’t update directly from the model, it always asks the controller for data. But the difference gets smaller as AJAX offers the possibility of polling the server for database changes, which creates a “direct” connection from the model to the view.

Besides controlling the result of the user actions and updating the model, the server is responsible for state management and creating the view, which are operations independent from the server, and that, as such, could be done on the client side, reducing the work of the server and
the database accesses, the typical bottlenecks. This is what is happening as client-side technologies become more powerful. We’re shifting away from the model where the server is responsible for the presentation and state management of an application and toward the model where the purpose of the server is to dish up structured data content.

### 3.4. Service Oriented Architectures

The distributed nature of Rich Internet Applications, especially in mash-ups, along with deploying to multiple platforms, stimulated the adoption of service-oriented architecture, i.e. an architecture where the back-end functionalities and business processes are exposed as autonomous functional units to be accessed through a network. There are lots of obvious benefits in this architecture, like encapsulation, loose-coupling and reusability. It is usually implemented through Web services that follow a protocol like SOAP [Wor03] to transmit XML data. [Erl04] This is the typical way how the servers dish up data to a RIA front-end.

### 3.5. Event-Driven Architectures

User interaction can happen at any time, and will have to trigger a function. Likewise, a request to a service can take an unpredictable amount of time to deliver the response that will be necessary to complete another function. These changes in state (up-down button and sending-waiting-receiving) are called events, and the scripting language can wait for these events and call the according functions. The typical event sequence, shown in figure 3.3., involves the registration of a function to a specific event, so that when this event is triggered the function will receive the event object as a parameter and run accordingly.

![Event handling sequence example](image)

**Figure 3.3: Event handling sequence example**
An event-driven architecture extremely loose-coupled components and functions (called event handlers), because the event can be any kind of object and carry any information necessary, and it doesn’t need to know its consequences or implications. Sadly and once again, this is one of the features not sufficiently developed in browsers’ interpretation of DOM Events [Wor084] and JavaScript, addressed by a standard yet to be followed, XML Events [Wor085].

3.6. Technological Requirements for Online Applications

The transition of the controller and state management to the client-side and the increase of complexity in online applications only started to happen in the last years. There were several technological limitations that delayed (and still delay) this changes:

- JavaScript wasn’t made for complex software development nor complex graphical interfaces;
- Since JavaScript was created mainly for acting over HTML, important concepts of object oriented programming (like encapsulation), as well as dynamic creation of non-HTML complex graphical interfaces were neglected. Even if these were addressed by the ECMAScript standard and others (SVG [Wor086], XHTML 2.0, ...), it is still not implemented in all browsers;
- Lack of standardization led to a greater amount of work deploying for all browsers than deploying to the desktop or having the logic running on the server;
- Bandwidth and interpretation processing time requirements;
- All the client-side code must be downloaded and then interpreted on the user’s computer;
- Keeping the state on the client;
- The importance of the Remote Scripting feature is that it allows the loaded page to get more data from the server and change its appearance accordingly, which changed the concept of the server-created pages that have to be reloaded with every action. Supporting custom event objects is also very important to allow complex event-driven stateful applications.

Technical Workarounds

Because of the browsers slow adoption of standards, the current solutions are based on complex JavaScript libraries and hacks, which solve most problems the browsers should at the cost of heavier loads, processing time and worst architectures (JavaScript doing XHTML’s or CSS’s work).

The other solutions involve embedded environments like the Flash [Ado08] and Silverlight players and Java Applets [Sun08]. These products offer a full EDA, multimedia and vector
graphics support and lots of other features that compete with browsers’. These technologies depend on closed non-standard formats and sandbox environments.

3.7. Post-AJAX Technologies

XHTML 2.0 [Wor07], Silverlight [Mic08] and Flex [Ado08] perform tasks that are very common and that previously were handled by scripting languages, (e.g. data validation and sortable views), and incorporate components that XHTML didn’t have, like video clips.

They also use concepts from common design patterns, and simplify their use. A great example is data-binding (exemplified in Figure 3.4.), which is a shortcut to the View’s dependency on the Model (in the MVC pattern that will be referred later).

![Data Binding Example in Flex](image)

Figure 3.4: Data Binding Example in Flex

They also include custom events, meaning that one can create new events that carry the information needed for a specific action, and not be restricted to the currently defined in the language (DOM events, in XHTML). This feature allows a full Event-Driven Architecture, which is a very powerful paradigm for interface development.

The big difference between XHTML 2.0 and Microsoft’s Silverlight and Adobe’s Flex is that XHTML2.0 was supposed to be implemented by the browsers (there isn’t much hope of seeing this in the near future, though; the existing implementations are plugins and JavaScript libraries). Flex is compiled into Flash and Silverlight is compiled JIT (just-in-time) and both run in closed source browser plugins. This means that the evolution of these platforms depends on Microsoft and Adobe, but it also means that the output is the same across browsers, something that was difficult to achieve with AJAX.

Furthermore, there are the differences in speed and error handling differences in interpreted (AJAX), compiled JIT (Silverlight) and pre-compiled into byte code (Flex).

Scripting (Actions) in Flex are written in ActionScript, the ECMAScript4-compliant language from Flash. In Silverlight, this is accomplished using Jscript, Microsoft’s JavaScript implementation. The Layout and Model in Flex is written in Adobe’s MXML, and Silverlight in Microsoft Windows Presentation Foundation XAML. The presentation (styles) is defined with CSS in Flex and XAML in Silverlight.
3.8. Advantages of XML-based Languages

Silverlight’s XAML, Flex’s MXML and XHTML 2.0 are XML, the data loaded into the application can be XML and it can submit XML. Besides providing consistency and encouraging code reuse, there are a number of advantages in using XML end-to-end [Hun07].

- There are lots of existing and proven XML technologies, such as XPath for addressing and calculating values, and XML Schema for defining data types. This has a dual benefit: ease of learning for people who already know these technologies, and the ability for implementers to use off-the-shelf components to build their systems;
- XML is device independent, i.e. the same form can be delivered without change to a traditional browser, a PDA, a mobile phone, a voice browser, or an Instant Messenger;
- It can represent some of the common computer science data structures: records, lists and trees;

```
<mx:ArrayCollection id="data" filterFunction="filter" sort="sortAlphabetically"/>
<mx:AdvancedDataGrid id="dataGrid"
                             dataProvider="(data)"
                             enabled="false"
                             selectedIndex="1"
                             allowDragSelection="true" />
```

Figure 3.5: XML-Based Flex MXML example

- Because of XML’s declarative markup ease to declare properties of values, and to build relationships between values, it is much easier for the author to create complicated, adaptive, forms, without having to resort to scripting. This simplicity is exemplified in figure 3.5;
- Text-based – can be created and parsed easily;
- It supports Unicode, allowing almost any information in any written human language to be communicated;
- The strict syntax and parsing requirements make the necessary parsing algorithms extremely simple, efficient, and consistent;
- It is a standard;
- It can be updated incrementally maintaining backward compatibility;
- It allows validation using schema languages such as XSD, which makes effective unit-testing, firewalls, acceptance testing and contractual specification and software construction easier.
3.9. Conclusion

The concepts in this chapter are not usually found in tutorials or technology documentation, but they are mandatory for a RIA developer and architect to know, and they should have a high impact on RIA design. From three-tier architecture to SOA, from dynamic pages to full applications, from standards to actual implementation, from scripts to AJAX and EDAs; the background on RIA is of extreme importance. An overview of the history of RIA is also valuable to a RIA developer, and it can be found in the end this document, under Annex A. This chapter provided key concepts for the next chapter’s objective, choosing the technology.
4. Existing Technology Comparison

Because of a current hype around RIA and its strong marketing and biased evangelisation, it is hard to have a clear notion of the strengths and shortcomings of a particular technology without a hands-on approach, and so many technologies were tried as thoroughly as the available time allowed. The previous two chapters were also part of the technology study, and provide a solid background for defining the parameters on which to judge RIA technologies. The technologies are similar in some of these parameters, but they were included nonetheless, because their similarity is not always clear in other available comparisons.

4.1. Important Considerations

The back-end of some of MOG’s applications was already developed in a services-oriented architecture, so it was important for the chosen language to have a SOAP implementation. There was a close deadline for some of products, so it was important that I could learn the technology fairly quickly, and the time necessary for others to learn it was also paramount. Finally, the portability of the developed software for web, desktop, and mobile was an important point as well.

4.2. Technologies studied

The field of studied technologies was reduced to the final three contestants: Adobe Flex 3 [Ado081], Microsoft Silverlight 2 beta 2 [Mic081] and AJAX [Hol08]. As noted in the previous chapter, the AJAX designation is used to describe the combination of XHTML, CSS, JavaScript (and JavaScript Libraries) asynchronously requesting XML. Not far behind though, it is very important to mention OpenLaszlo [Las08], XHTML2.0 and JavaFX [Sun08]. OpenLaszlo is an XML-based language with most of the features the other best RIA languages offer, with the particularity of being converted optionally in both Flash and AJAX. It is not as widely adopted, though, and the community size, available tutorials and blogs describing common shortcomings and solutions were the comparison points that made it come in a close fourth place. XHTML2.0
Rich Internet Applications – Study and Development

had the advantage of being the open standards solution, but it is only implemented in plugins and JavaScript libraries which evolutions were less trustworthy. JavaFX is the very promising non-XML-based alternative from the Java world, but it is still in a pre-alpha state.

4.3. Comparison

Reach

The technology reach study tries to predict the availability of an application for multiple environments and different kinds of users. Table 4.1 shows how the selected technologies compare in this area.

<table>
<thead>
<tr>
<th>Browser and Operating System compatibility</th>
<th>AJAX</th>
<th>Silverlight</th>
<th>Flex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every browser and, consequently, every OS. But there it takes a lot of work to make sure it is rendered the same way in every browser</td>
<td>Top 4 browsers. Mac OS-X and Windows. Project moonlight, currently in development, will port it to Linux.</td>
<td>Top browsers and OS</td>
<td></td>
</tr>
</tbody>
</table>

| Mobile and desktop deployment | Adobe AIR allows AJAX, Flash and Flex applications to be deployed on the desktop (Linux support in beta version). Prism is a technology that allows like a browser window Customization is required for mobile browsers | In the near future, Silverlight applications will run on Windows Mobile. Since Silverlight is a subset of WPF, it shouldn’t be hard to change a project to deploy on the desktop | Adobe AIR to deploy on desktop, Tamarin Tracing and Flash Lite will bring Flex to the mobile world in the near future |

| Ubiquity | Every Browser | Still very rare, but might be distributed with Windows or through Windows Update in the future | Flash player is in more than 97% of the browsers [Mii08] |

| Installation ease | Only a browser is needed | Easy and Fast | Easy and Fast |
| Start-up time | Just the browser loading time | Fast | Fast |

Silverlight is still not widely spread and its Linux or non-Windows Mobile support is neglected. AJAX requires a lot of work to assure it is rendered correctly in different environments.
Trust

Investing on a technology is a difficult commitment to make in an always changing environment like RIA’s. It is important to trust the continuity of the chosen technology. Taking subjective bias about the companies aside, the trust comparison is shown in table 4.2.

Table 4.2: RIA Trust Comparison

<table>
<thead>
<tr>
<th></th>
<th>AJAX</th>
<th>Silverlight</th>
<th>Flex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maturity</td>
<td>Very mature, with lots of problems to solve but many known workarounds as well</td>
<td>Very young, not much experience, but very well designed</td>
<td>Mature and well designed</td>
</tr>
<tr>
<td>Known adopters</td>
<td>Thousands of companies</td>
<td>Most of the people currently developing are under Microsoft’s financial incentives</td>
<td>Many</td>
</tr>
<tr>
<td>Age</td>
<td>very old (see History of RIA for particular dates) with the exception of particular toolkits/libraries</td>
<td>about a year</td>
<td>about 4 years</td>
</tr>
</tbody>
</table>

All the three choices have a great commitment and a fast improvement rate, but Silverlight still has to prove that it’ll succeed.

Adopting Costs

To adopt one of these technologies requires an investment in time and money. The size of the investment is the scope of table 4.3.

Table 4.3: RIA Costs Comparison

<table>
<thead>
<tr>
<th></th>
<th>AJAX</th>
<th>Silverlight</th>
<th>Flex</th>
</tr>
</thead>
<tbody>
<tr>
<td>License</td>
<td>Open source</td>
<td>Proprietary and closed</td>
<td>Open source SDK</td>
</tr>
<tr>
<td>IDE Price</td>
<td>Free</td>
<td>~$500 for Visual Studio or Expression Blend</td>
<td>~700 Flex Builder</td>
</tr>
<tr>
<td>Integration with existing code</td>
<td>Web services</td>
<td>Silverlight applications can be developed using .NET languages (so far, Jscript and Python, Ruby and VB in the future)</td>
<td>Web Services</td>
</tr>
<tr>
<td>Languages and development paradigms</td>
<td>Many languages to learn, many problems to avoid</td>
<td>Well designed language, incorporating MVC and EDA. Having the choice of the scripting language is a plus</td>
<td>Well designed easy to learn languages, incorporating MVC and EDA Languages only used in Flash world</td>
</tr>
<tr>
<td>Learning resources</td>
<td>Very strong developer community, hundreds of blogs, forums and tutorials</td>
<td>Weak community, but growing fast. The main forum has quick answers</td>
<td>Strong develop community, two main forums with next-day answers, many blogs and excellent documentation</td>
</tr>
</tbody>
</table>
Rich Internet Applications – Study and Development

Flex Builder is expensive, and the free alternatives are not as good. AJAX is the most difficult to learn of the three technologies, but it is well documented. Silverlight support of all .NET languages allows using previous experience.

**Development Ease**

The most important cost is the continuous cost of development time. Table 4.4. compares RIA technologies in this parameter, which is very important because the technology will not be used on a single project.

<table>
<thead>
<tr>
<th></th>
<th>AJAX</th>
<th>Silverlight</th>
<th>Flex</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Existing Libraries</strong></td>
<td>Lots of toolkits, usually a combination of which is required.</td>
<td>Nothing relevant yet</td>
<td>Many interesting projects (See the development environment chapter)</td>
</tr>
<tr>
<td><strong>Existing components</strong></td>
<td>Toolkits (e.g. Ext JS)</td>
<td>Very few</td>
<td>A lot of default components and many community-developed components</td>
</tr>
<tr>
<td><strong>IDE</strong></td>
<td>Several good IDEs (e.g. Aptana)</td>
<td>Visual Studio (Development) and Expression Blend (Design), both very good tools only for Windows</td>
<td>Flash IDE, Flex Builder and Eclipse plugin; very good tools</td>
</tr>
<tr>
<td><strong>Unit testing, debugging, code completion</strong></td>
<td>Yes</td>
<td>Unit testing is possible with some tricks (faking a WPF application) Debugging and code-completion supported.</td>
<td>Unit testing community library. Debugging included in IDE</td>
</tr>
<tr>
<td><strong>Code Size</strong></td>
<td>Large and ugly</td>
<td>Average, many files and few components</td>
<td>Reduced</td>
</tr>
<tr>
<td><strong>Extensibility and customization</strong></td>
<td>CSS, JavaScript, limitations</td>
<td>Very easy to extend and customize</td>
<td>Easy to extend and customize presentation, a bit harder to customize shapes or effects of parts of a component. This can be accomplished with Degrafa and Openflux frameworks, and will be addressed in the next version of Flex</td>
</tr>
<tr>
<td><strong>Porting to desktop application</strong></td>
<td>Easy</td>
<td>Several care needed converting Silverlight to WPF and file structure</td>
<td>Very Easy</td>
</tr>
<tr>
<td><strong>Accessing the system (desktop)</strong></td>
<td>Easy to access the file system with AIR. Fluorine Aperture hack for executing commands</td>
<td>Easy with .NET languages</td>
<td>Easy to access the file system with AIR. Fluorine Aperture hack</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Widely known and solvable problems</td>
<td>Complex sandbox environment</td>
<td>Somewhat restricting sandbox environment</td>
</tr>
<tr>
<td><strong>Bugs</strong></td>
<td>Many Bugs</td>
<td>Not old or stable enough to know</td>
<td>Very few bugs</td>
</tr>
</tbody>
</table>

Table 4.4: RIA Development Ease Comparison
Silverlight has a very good language paradigms but not many components and libraries. AJAX has the worst languages because of browser support limitations. Adobe AIR is the best solution for the desktop, since it requires almost no changes to the code for web deployment. Flex is strong on development ease.

**User Experience**

The quality of user experience is hard to measure, but it is possible to identify some key technological requirements that allow the most common interface design patterns, shown in table 4.5.

<table>
<thead>
<tr>
<th></th>
<th>AJAX</th>
<th>Silverlight</th>
<th>Flex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default look-and-feel</td>
<td>Document look</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Customizable look</td>
<td>CSS-based</td>
<td>Powerful XAML-based customization</td>
<td>CSS-based. It is also possible to import flash assets. In the next version of Flex, there will be MXML-based customization, in the likes of Silverlight's</td>
</tr>
<tr>
<td>Vector Graphics (resolution independence)</td>
<td>SVG is unlikely to be supported in the future</td>
<td>Good XAML support</td>
<td>Excellent AS support</td>
</tr>
<tr>
<td>Animation and interactivity</td>
<td>Possible with toolkits (dojo, processing.js) but slow and not very complex.</td>
<td>Somewhat overly complex storyline concept, but powerful nonetheless. Deep Zoom offers the possibility of Zoomable interfaces</td>
<td>Excellent AS and MXML support</td>
</tr>
<tr>
<td>Crawler-enabled, backbutton, deep-linking support and accessibility</td>
<td>Content is searchable, but there are many accessibility problems, addressed by the WAI-ARIA specification and some known tricks. Some toolkits (e.g. Dojo) already follow WAI-ARIA</td>
<td>Not many solutions yet. Because XAML is XML compiled JIT (unlike a compiled .swf), it will probably be supported in the future</td>
<td>It is possible to create searchable, back-button enabled and accessible applications, but it requires additional work</td>
</tr>
<tr>
<td>Video and audio support</td>
<td>No support, embed Flash or Silverlight player</td>
<td>Very Fast</td>
<td>Fast and mature H.264 support</td>
</tr>
<tr>
<td>Performance</td>
<td>Fast for simple pages</td>
<td>Faster [Ale08]</td>
<td>Fast</td>
</tr>
</tbody>
</table>

Lack of browser support for standards makes AJAX the worse tool for complex graphics and multimedia. The plugin-based approaches are less accessible, with provides worse integration with the browsers and search engines.
4.4. Conclusion

Table 4.6 summarizes the analysis in this chapter, with a "+" meaning a strength and a "−" a weakness in each particular field.

<table>
<thead>
<tr>
<th></th>
<th>AJAX</th>
<th>Silverlight</th>
<th>Flex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reach</td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Trust</td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Adopting costs</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development Ease</td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>User Experience</td>
<td></td>
<td>−</td>
<td>+</td>
</tr>
</tbody>
</table>

All three technologies have great qualities, and there isn’t an obvious winner for all scenarios. It is a very complex choice. But by prioritizing features for a given project, it becomes easier to choose. These were the most important points that conditioned the choice of RIA technology for MOG:

- It is hard to believe that Microsoft will care about non-Microsoft platforms. Porting an application to the desktop still requires more work than with Adobe AIR. Silverlight is still young and unproven;
- AJAX has the poorest look and media support. It takes a lot of time and effort to become a good AJAX developer. JavaScript browser support has a lot of flaws;
- Flex is easy to learn and to develop with, without many bugs and very well documented;

Flex was chosen as the technology to be used in many products, and by many of MOG’s developers. The next chapters are related to Flex development.
5. Flex Working Environment

Having chosen the technology to use, it was important to learn it well and understand how it could be used most efficiently. The information found in Flex’s documentation [Ado082] is a good starting point, and there are good several books to complement it [Cas07] [Tap081]. Flex.org provides links to most of the interesting sites to continue learning from or to download community created components.

Throughout the project, there was a search for good practices that could further improve the development of Flex RIA. The result of this study is presented in this chapter, and these concepts although not obvious are simple to understand and extremely useful for any RIA developer.

5.1. Implementation Patterns

During the design and development phases, I struggled with some recurring questions about architecture. Applying known design patterns [Mar08] [Gam94] to the questions in hand was often a good approach. In RIA interface development it is common to find clumsy and chaotic code, because of the difficulties in adapting JavaScript to object oriented patterns and the less programming oriented education of designers. Flex is much tidier but it is still easy to fall into some mistakes. This chapter includes some considerations that promote code reuse, agility, and easy to read code. It also describes some solutions to Flex specific shortcomings.

When to use MXML and ActionScript

It is possible to develop a simple Flex application solely in MXML. The components allow inline event handling and binding to other components. It is also possible to write everything in ActionScript, but MXML adds a lot of architectural and coding benefits. As applications get complex, it becomes obvious a careful mix of the two languages should be used. Oversimplifying, MXML is good for views and bindable models; ActionScript is good for event-handling and complex data structures. Containers and components should be declared in MXML, and so should their states and effects. The events fired, though, should always be handled in ActionScript functions. Data classes and non-visual components should be coded and instantiated in ActionScript as well. The exception is data objects which parameters directly depend on other
objects might be instantiated in MXML, to take advantage of data binding. A good example is when some text input fields determine attribute values of an XML that will be sent to a web service, as shown in figure 5.1. The XML might be declared explicitly and bound to the interface component.

```xml
<mx:XML>
  <person>
    <first_name>(firstNameInput.text)</first_name>
    <surname>(surnameInput.text)</surname>
  </person>
</mx:XML>

<!-- when the user enters his name, the xml is automatically updated -->
<mx:Form>
  <mx:FormItem label="Please enter your name" />
  <mx:FormItem label="First Name">
    <mx:TextInput id="firstNameInput" />
  </mx:FormItem>
  <mx:FormItem label="Surname">
    <mx:TextInput id="surnameInput" />
  </mx:FormItem>
</mx:Form>

Figure 5.1: Declarative XML instantiation and data binding
```

Declaring an object in MXML has the advantage (or disadvantage, when some of these functions are not desired) of declaring, instantiating, making it bindable, setting properties and importing necessary classes all at once. Figure 5.2. shows the way MXML and ActionScript do an equal task.

```xml
<mx:ArrayCollection id="data" filterFunction="filterEven" sort="sortById" source="otherdata" />
```

```java
import mx.collections.ArrayCollection;
[Bindable]
public var data:ArrayCollection=new ArrayCollection();
...
data.filterFunction=filterEven;
data.sort=sortById;
data.source=otherdata;
```

Figure 5.2: The same object created in MXML and in ActionScript

**Code Structure**

Flex applications are a usually a tree of components. As said before, as applications get more complex, separating code by its different functions becomes more important, for maintainability and readability purposes. The example in figure 5.3 proposes a separation of concerns for code-heavy components.
<Metadata>
   <!-- declaring events the component dispatches and other Flex compiling options -->
   [Event(name="refreshEo", type="mx.events.ItemClickEvent")]
   ...
</Metadata>

<Services />

<object>
   <!-- typically dataProviders and model -->
</object>

<!-- if the component is complex, use source attribute to refer an .as file that contains the code -->
<fx:Script source="?"/>

<![CDATA[
 import ....
 ...

 // declare static
 public static const MY_EVENT:myEvent = "myEvent";
 // other objects
 public var ...
 ...

 public function createComplexHandler():void{
   // add all event listeners here
   button1.addEventListener(...);

   // whenever an event handler can be reused,
   // put it in a static function of a helper class
   myComp.addEventListener(... EventHandlers.eventHandler);
   ...

   // typed event handler functions
   public function myEventHandler(event:CustomEvent):void{
      ...
   }
   ...
]]>

</fx:Script>

<Bindings, formatters and validators i.e. model-view connection -->
<Binding destination="..." source="..." />
<mx:DataFormat id="dataFormat" />
...

<Validators>
...

<Graphical components without styles, bindings or event handling/>

<!-- if the component presentation is complex, use source attribute to refer a .css file -->
<fx:Style source="?"/>

<!-- states, effects, transitions -->
<mx:States>
...
<em:Discrete -->

Figure 5.3: Suggested code structure
Mediator

A Flex application is a tree of components and containers. There are many possible ways to connect these components. The mediator is a class that makes this connection, allowing the components to be completely loosely-coupled and know nothing about each other. It is a good practice to have components fire events and to fire events into a component, in a complete event-driven architecture, rather than binding them together. Custom events can carry complex structures, making it possible to aggregate and generalize events.

![Diagram of the mediator pattern in Flex]

Figure 5.4: Example of the mediator pattern applied in Flex

In the example of figure 5.4, the shopContainer is a mediator for the itemList and shoppingCart components. Instead of binding a shoppingcart.add2SCart(item) function to itemList.selectedItem, itemList just fires an event containing the item. This event is caught by the mediator, which in turn fires an event into the shoppingCart component that is listening for such
events. The Application container could also serve as a mediator for shopContainer, componentX and componentY.

When refactoring, every time a container has more than one component in it, it is a good idea to think if it would make sense to make it a custom component (in its own separate class).

**MVC**

The Model-View-Controller pattern has been referred multiple times in this document. The idea behind this pattern is to separate the data and state (model), from the business logic (controller) and the presentation for clarity, simplicity, reuse and making future changes easier. In Flex, separation the services is also valuable. Although a simple idea, it is common to see this pattern being broken, which shouldn’t happen without a reason. This pattern can be applied to the architecture of an application as well as of a custom component. An example of the MVCS (“S” for Services) is presented in figure 5.5.

![Diagram showing Model, View, Controller and Services](image)

**Figure 5.5: Model, View, Controller and Services**

**Field encapsulation**

It is common to declare a class attribute as private, and to create mutator (setter and getter) methods. In Flex, a setter method can be accessed like if it was an attribute, rather than a method. More than just syntactic sugar, this allows these attributes to be set declaratively (in MXML) and bindable.

The other methods of the class can access the private attribute, but in Flex this is not recommended if the attribute’s getter is bindable, since it is the setter method that dispatches the change event that allows bound object to update their value according to the new value returned by the getter.
Modules

The only reason to develop using modules is being able to control when some part of the code is loaded. This is important for very large web applications, which would otherwise consist of a single huge .swf. There are some caveats when using modules, thought:

- There is a bug with views stacks (tab navigators, link and button bars) in modules which is not documented. The history manager (which handles navigation and deep-linking) instantiation is assigned to the first module loaded, and a runtime error occurs when a second module tries to make use of it. The workaround involves creating a new history manager on the module load event, and handling it “manually”;

- Debugging is not supported for modules, because the debug versions of the .swf files in are different from the release versions the Flex automatically builds when using modules. Knowing this makes it easy to solve this problem, since you can force the module loader to use the debug version. One can even create a static function that checks whether the main application is running in debug mode or not, and choose return the .swf url accordingly. This solution was implemented in a reusable component described later in this document.

Flex web services generation

Flex Builder 3 provides automatic generation of Service classes that provide the methods defined in a given WSDL or WSDL location. It also creates typed classes for the service methods results. Every time the service methods change, one can regenerate these classes. While this automation is useful, there are two limitations in using the generated classes: their parameters aren’t bindable and there isn’t a function to change SOAP headers. It is not difficult to change the classes to add this functionalities - add the [Bindable] tag to the class and change the instantiation of the header arrays in each web service method in the Base class from “new Array()” to the global variable headers (and change headers when needed). The problem is the next time the classes are generated the changes will be overwritten. The better alternative is to extend those classes and override them to solve the limitations.

When polling a service with a small interval (< 0.1s), the event listeners involved in handling service answers are not properly garbage collected and there is a memory leak. This is not a commonly referred leak because Adobe also offers LifeCycle Data Services and BlazeDS, which are two versions of server-side complements to Flex that allow great communication improvements. The only possible workaround is to have the service run in a module, with its connection to the model included, and then have this module reloaded when the memory reaches a certain ceiling.
Dynamic changes to the layout

Flex allows containers to automatically resize with the application window or browser by defining its width or height as “100%”. Another useful feature of Flex containers is the layout attribute. When the layout="horizontal", for example, the components inside the container are automatically positioned sequentially from left to right, as opposed to an absolute layout where they would be overlapped, or with their fixed position defined by “x” and “y” attributes. These features, while useful for fast development, have performance costs, and an absolute positioning and dimensioning should be preferred when possible.

Four different approaches were used to allow changes in the interface layout during runtime:

- States are good for small changes, like changing some styles or hiding a single component, since a state is defined by its differences to another. The state concept is very good, but the syntax is a bit too verbose. This is one of the few where Flex Builder’s Design View comes in handy;

- The lowest level approach is to do it programmatically with ActionScript addChild(), removeChild(), setStyles() and other such functions. This was only used when unavoidable, because it is not hard to lose control if one is changing the layout in several functions. The most common operations are available in states, but when AS is needed, it is preferred to define all changes related to a state in AS rather than using a combination of states and AS;

- Popups allow maintaining a state visible (or partially visible, possibly blurred) in the background while in a new state. Flex’s PopupManager allows any component to be a popup. This is a good interface design pattern for configuration and submittable forms, since after submission, the application goes back to its previous state;

- When the interface changes to a complete new set of components, view stacks should be used. View stacks are the most straightforward solution, its code being very simple and organized.

In all these cases it is important to be careful about binding to and using components not instantiated. A possible way to avoid such problems for small or desktop application where loading time is not crucial is to force the instantiation of all objects from the start and to change their visibility instead of removing and adding them. This is also offers considerable performance improvements, as changing visibility is less time consuming then adding and removing objects.

Frameworks

There are several frameworks that weren’t used in the applications developed during this project, but that are worth mentioning. PureMVC [Fut08] and Cairngorm [Ado083] are mature MVC architectural frameworks. As any architectural framework, they reduce problems to a set of patterns, which might end up in providing more complex solutions than necessary. The upside is that the proven patterns will be followed.
Flex components are not decomposable in parts, which makes it difficult to remove and combine their features and look. Openflux [Stu08] is a components framework that includes copies of Flex components decomposed in an MVC manner, allowing a lot of extra flexibility. Degrafa [Sca08] is a framework that allows defining graphics in MXML (bindable and reusable) as well as applying transformations and repeaters among other features and is good for Openflux's views. Some of the features Openflux and Degrafa offer might be included in Flex 4.

5.2. Flex Development Workflow

There were some practices at MOG Solutions to adapt to. The builds were automated, logging followed an internal standard, bugtracking was accomplished using Bugzilla [Moz08] and revision control using CVS. For my last project, MXF SpeedRail Metadata Editor, SVN was used instead of CVS and Trac [Edg08] instead of Bugzilla.

Since choosing Flex as MOG's RIA technology had a high impact (more than a third of MOG's developers started using Flex in their workflow), it was important to define guidelines to be followed when developing in Flex.

At MOG, new requirements come at unpredictable times and changes are very common. Agile methodologies were used as an inspiration in many of the choices. For small projects some of these practices might not be taken, but not using them should be a conscious choice, and not a uninformed risk. Although not having followed all these steps for every project because the tools required were not yet available, these are the steps recommended for a standard implementation cycle.

Feature Requests and Bugtracking

MOG had already chosen technologies for feature requests and bugtracking, Trac and Bugzilla, which are great solutions for these tasks. The Mylyn [The08] eclipse plugin integrates with Flex Builder and does a good job in integrating feature requests with Flex Builder. Mylyn is a great plugin, not only for its integration with bugtracking and project management software, but also for its incredibly intelligent task management functionalities.

Layout Specification

Prototyping the layout of the application is something that can be accomplished rather easily in Flex. It is also one of the most difficult requirements to think thoroughly and describe verbally. During the requirements meeting, a sketch of the interface would be created, and the first thing done after the meeting would be turning it into Flex, which is possible in about ten minutes. There wasn't a single project where the interface requirements didn't change after seeing the ideas implemented.
Test cases

It is fairly easy to turn requirements into tests with Flex Unit [Sch08]. Test-driven development was only used for high level requirements and not for each implemented function, an approach that is recommended when requirements change often. Although tests are incredibly valuable when changing a complex system, one should avoid writing tests that couple the tests with the implementation rather than the feature, at the price of having to rewrite tests every time class properties change. Data-centric tests should be post-poned to the refactoring phase, with the purpose of make future changes reliable, rather than using them as a specification of what to implement.

There are many tasks that are hard to include in unit test in interface development, especially user interface requirements. There is a Flex visual testing framework [Ber08] that allows testing an interface against a picture of the required look. This is especially useful when the look of an application is defined by a designer and later implemented by a developer, which was not the case of the projects at hand. The separation of controllers, event handlers and state handling into separate functions is paramount for proper testing, as inline handlers can’t be called by test cases.

Creating Dummy Services

Working in a Service-oriented architecture where the back-end and the interface are developed simultaneously, it is common not to have the final version of the service one is using. Furthermore, a simple network failure can compromise the nature of the project. Therefore, it is important to be able to run the interface independently from the backend, and such was accomplished by creating dummy models of the data to be passed by the services. Flex’s declarative instantiation makes models so readable that they were put in a separate file and used a specification artefact.

Writing code

The actual writing of the code was mostly done solo, with the exception of the development of XDCAM Controller and EDL Converter, during which I pair-programmed with a co-worker who was learning Flex. Pair-programming is equally valuable in Flex as it is in other technologies, and although this document will not go into depth discussing this practice, it highly encourages studying where (and who) it fits best.

As standards are adopted, a greater part of the code becomes similar in every component. This allows code generation. A good example is the code structure or mutator methods. Code snippets speed coding, and were extensively used. Another plugin that proved useful was the Fix me/ todo plugin that provides a list of parts of the code that still require attention (marked by the developer or included in code snippets).

Adobe AIR allows accessing the file system quite easily, but there isn’t a way to execute other applications using the command line. This was done using a very intelligent hack called FluorineFX Aperture [Flu08].
Debugging

Although flex has built-in logging capabilities, an improved logging system was created in order to simplify the process and incorporate the company standards. When the bug is found by a user in a hard to describe use case, logs are the best way to try to recreate the situation and find the bug. Testing the SOAP services was accomplished with SoapUI [Evi08] and debugging services requests and answers was done with a sniffer software called Charles [von08], two very valuable tools.

Version Control

To ensure the work is never lost and changes are undoable, version control is very valuable. Flex builder supports CVS out of the box and the Subclipse [Col08] plugin was used for SVN.

Automated Builds

Integrating Flex in an automated building process is fairly easy, since there is a command line compiler, and its use for desktop applications, web and modules and compiling options are well described in the documentation. The only caveat is one should have the Flex Builder Professional licenses installed in the computer that runs the build script; it is not enough to run the professional compiler to get the professional features. Deploying to the desktop involves altering an XML file that has some configurations for the application, which can easily be done with a script. When deploying a web application, it is important to have a standard path to the folder where assets that aren’t compiled in the .swf are kept, so that it is copied to the release installer, and a HTML and JavaScript wrapper that calls the .swf file.

Refactoring

When seeking performance measurement and improvements Flex Builder’s profiler is the best tool. It provides detailed description memory and processor usage and the objects instantiated at any debug time. The design patterns described in the first part of this chapter help in searching for architectural improvements. Other code structure improvements can be searched looking for unnecessary code dependency with ItDepends [Berk08]. Test coverage was measured after the features were implemented, and additional tests were added where necessary. This was done using FlexCover [Berk08].

Architectural refactoring was sometimes abused, because it was also used as a way of exploring and learning Flex. Using tests and version control, refactoring is simplified, since it is possible to verify the changes didn’t alter the functionality, and if they did, roll back to the stable version.

Documenting

Even if an application is supposed to be used by a single coder it is useful to have bits of the code commented and the class properties listed and described in a readable way. ASDoc is integrated in Flex and it is a valuable tool to use.
**Manual Testing**

Features and functional requirements list was checked, and the application was tested by users that had never seen the application. A fixed set of questions were then asked to these users about the look-and-feel and interaction ease of the applications. The recreation of the use cases was timed. Code coverage during tests was also measured with FlexCover.

### 5.3. Future work

Defining and documenting good practices is in itself a very good practice, and further improvements are always possible. The next areas where an in-depth study is lacking in the Flex environment are adapting UML (simplifying, actually) to the Flex development workflow. There are UML tools that allow a lot of code generation from UML diagrams, and if a reduced and more adequate to RIA technologies diagramming standard was defined, and a Flex/ActionScript code generation module added to a UML tool, the job of a Flex solution architect would be greatly enhanced. This is obviously a demanding task and a costly investment, unlikely to be developed internally in a company where Flex is not the main technology used.

### 5.4. Conclusion

In this chapter, known patterns from software engineering were applied in RIA development, setting a strong environment for any future development of Flex applications. Solutions to simple recurring problems were explained and serious coding and design considerations that should be on any Flex solution architect mind were explained. Finally, a full development cycle was described, with phases that may or may not be included in a project depending on its characteristics, introducing an extensive list of tools and libraries available that help each phase.

The ideas in this chapter are the result of both study and development phases of the project. The next chapter is about the latter.
6. Developed Applications

This chapter will provide a description of the implementation part of this project. Like in the rest of the document, an effort was made to separate the reusable patterns, components and lines of thought in the project, avoiding meaningless details overly technical, technology specific or low-level. Because of the large number of applications and components developed, a succinct description and free choice of implementation details is provided for each, with the objective of showing the high productivity a well thought Flex development workflow offers and the usage of some concepts referred in the previous chapters.

6.1. Reusable Components

There were parts of the applications that, because they solve common problems, were designed in separate components, generalized for reuse. These components are described separately in this section.

Submit Popup

One of the recurring interface patterns was the configuration popup window. A user wishes to configure an application, clicks a button saying configure, the application blurs and there is a popup with a certain number of fields to be filled or changed and then returned to the main application. This highly customizable component was created using a class that creates the popup and stores its return object(s) and allows any type of input (date, text, checkbox, enumerations) and validation. Although some of the ideas in this component are similar to the XMLEasyConfig, the content of a component is defined declaratively instead of dynamically, allowing lots of performance optimizations.

XML Navigator

A tree is simple way to represent hierarchical data. In XML the parameters names are defined either in the element tags or in the attributes’ names, and the parameters value is between tags or in the attributes’ value. The XML Navigator component is a way of representing this, using a tree on the left side of a table and the value on the right side. The node labels and the values
shown can be defined easily, with options such as “data between nodes” or “data in attribute values” and label functions that remove underscores or CamelCase. The next picture, figure 6.1., shows two different XMLs, that can be rendered in the same way using different configurations (inputFormat="camelCase"/"underscore"; valuesLevel="attribute"/"node") of the XML Navigator component.

```
<input>
  <c001>
    <in_point_timecode>0</in_point_timecode>
    <out_point_timecode>-1</out_point_timecode>
    <relative_timecode>True</relative_timecode>
    <comments></comments>
    <tape></tape>
    <disk_label></disk_label>
    <type>
      <type1></type1>
    </type>
  </c001>
</input>
```

**OR**

```
<input>
  <c001 inPointTimecode="0" outPointTimecode="-1">
    <relativeTimecode="True" comments="" tape="" disklabel="">
      <type typel="" />
    </type>
  </c001>
</input>
```

Figure 6.1: Two different XMLs rendered in XML Navigator

**Connection Status**

When polling a web service, it is important to give feedback to the user about the connection status. The connection status component calls a supplied polling method with a user-specified frequency, reports the time of the last answer and announces when the connection is lost, offering the option of retrying to connect. Both states of the component are shown in figure 6.2.
some additional control when loading multiple modules, such as sequential or event-triggered loading.

**Data classes and libraries**

Some of the data classes are used in several of MOG’s products. A COD (Crystal Operation Description) is the standard MOG Toboggan Operation descriptor, with many details about a given operation. It is used in the whole line of Toboggan products, described in the next section. The COD class offers many of the attributes and a constructor from a COD XML.

A timecode defines a frame in a video in the format “hour: minute: second: frame”. To define a section of a video, to timecodes are selected, called marks: mark in defining the beginning of the section and mark out the end. The frames per second and the numbering of the sequence of the frames varies in different systems (PAL, NTSC, SECAM). Timecodes and marks classes were created that offer validation, comparison and other useful common tasks. Timecode and mark input components were also developed for MXF SpeedRail.

Finally, ActionScript lacks some functions that were implemented in static methods of utility classes (like cloning objects, flattening arrays, parsing date strings, etc.).

**Item renderers and editors**

In list based components (data grids, trees, tile lists, etc.) each item in the data provider is rendered as a string and, if editing is enabled, editable in a text input box. It is possible to use customized renderers and editors. Several item renderers and editors were developed (e.g. the status column in Toboggan Monitor or many of the components in XMLEasyConfig) with some advanced performance considerations, and compiled in a library for future reuse.

### 6.2. Other ideas for language enhancements

The advantage of Flex being an open-source project is that many of the developed components can be proposed to be included in the source. The reusable components developed tried to fill gaps in Flex, and although it is very good to be able to build components around the framework, some of these ideas could be integrated in the SDK. This section describes other such ideas not implemented yet, but for which the solution has been designed.

**Event component**

Since there are lots of event handlers that are instantiated when a component is created and never removed, it seems obvious there should be a way to define an event in MXML. This would allow the definition and binding of multiple event handlers, and the useful “enable” attribute and “dispatch” method.
2-Way-Bindable XML/Model/Object Components

A common scenario is having a form which input is supposed to be stored in several attributes of an object, which can be done as exemplified in picture 5.1. It is also possible to have an input field value depend on a single object attribute. It would be useful to have an option for two way bindings to do both simultaneously, allowing to declaratively bind to and from several attributes of an object (for the example at hand, by the way, it would also be useful a dataProvider field on forms). The designed solution involves parsing the object component children and creating bindings at load time or, in the case of XML objects, finding binding patterns ("{w}") and binding the data setter to the referred variables (similar to how the XML constructor is implemented in Flex SDK). These ideas, together with the observer pattern described in the reusable component with the same name, would improve the already very useful data binding.

Delayed instantiation

As referred in the implementation patterns section of this document, there are times when an object property is accessed without it being instantiated, due to states, views or module use, and a runtime error is thrown. It would be useful if controlling their instantiation as well as providing mechanisms for referring them when not instantiated, like it happens with data bindings.
6.3. Toboggan

Toboggan is a set of automation tools for professional video. Many common tasks like converting video formats, distributing video to specific devices and editing stations, editing, copying and annotating videos can be scheduled and triggered by events such as a video disc being inserted in a deck. Because of Toboggan's many uses and the importance of performance, a distributed architecture is required. The architecture is very modular and is service-oriented.

![Toboggan backend Architecture](image)

Figure 6.4: Toboggan backend Architecture

The typical Toboggan application has a graphical interface where the user configures one or more operations to be automated upon an event (usually a file landing on a specific location or device), scheduled or performed right away. These configurations and orders are forwarded via SOAP to a Toboggan Controller in a generic operation format, a COD (Crystal Operation Description). The Toboggan Controller then dispatches the operation to an available processing resource (Toboggan Core), commanding the flow of the operations. Most of the operations involve converting to or from an MXF file, wrapping/unwrapping a video in an MXF file, and performing operations on MXF files. MXF is the leading standard container format for professional video and audio. Each of the several different MXF profiles contains not only the media essences, in their various codecs, but also the metadata associated with it. MOG are international MXF software leaders, and all its products are related to MXF.
6.3.1. Toboggan Monitor

Toboggan operations follow a common interface, all of them providing some functions like status checking, cancelling, or retrying. All Toboggan products also allow the creation of different processing chains that may handle different operation or have different output locations. This common functionality offered in Toboggan Monitor, which is part of every Toboggan product.

*Use Cases*

![Diagram of Toboggan Monitor use cases]

Figure 6.5: Toboggan Monitor use cases
Requirements

- Modularity – This component is supposed to be integrated with many desktop applications, e.g. EDL Converter and XDCAM Adapter, as well as a standalone web application;
- Generalization – The possible types of processing chains and possible operations change according to the application that is using Toboggan Monitor;
- The operation details format is returned from a service in an XML that might change over time, and according to the allowed operations;
- The IP and port configuration will be done in the main application, so it should be allowed to update it on the background;
- Allow manual reconnect when connection is lost;
- Simple, minimalist and intuitive interface;
- High performance.

Solution Details

The Toboggan Monitor is a component that is deployed in a web application, desktop application or a module to be included in other Flex applications. For all these scenarios only a simple wrapper is necessary, "<mx:Application>" , "<mx:WindowedApplication>" or "<Module>", respectively.

The main monitor component can be configured passing it an XML at runtime or changing a configuration file before loading. The XML determines the Toboggans to be monitored, and each toboggan will have its own tab at the top and its respective monitor. Figure 6.6: Monitor Configuration XML example offers an example of such XML files, defining the logging configurations to be followed by the logging component as well as the location of two Toboggans to be monitored. The status of the connection to a Toboggan is handled by the Connection Status component described in the reusable components section.

Multiple operations can be selected (with Shift or Control keys) and retried/cancelled. The status and progress bar are shown in the same column - The progress is shown when status="running" and when an operation is completed the completion time is added.

The Operation details of the first of the selected operations are shown in a resizable area below the operation list, which can be hidden if not necessary. The details are rendered from an XML answer from the Web Services using the XML Navigator component described in the reusable components section.
The configuration of processing chains is accomplished using a component in another view stack that lists available processing chains. Adding/removing a processing chain opens a configuration Submit Popup described in the reusable components section. If the list is empty, a default processing chain is created, according to the Toboggan Monitor configurations.

**Implementation Patterns**

Figure 6.7: Toboggan Monitor Implementation Patterns depicts a few chosen details of Toboggan Monitor implementation, to demonstrate how some the practices described in the previous chapter can be incorporated in an application. In the application separation of model (orange), view (blue), controller (purple) and services (green), the omitted controller would consist of the event handlers that react to user input, perform general operations and connect the all the components (model, view or service). The connection between the view and the model is done with data binding, the service updates the model and the communication view-to-view and view-to-service is made through the controller, using the mediator. The selected operation details are rendered using the XML Navigator component, which can be visible or hidden, and this small change is done using states. A major change such as going to the completely different processing chains screen is done with view stacks. The submit popup component is used to provide a configuration screen while maintaining the processing chains semi-visible. The generated classes are extended and overridden to allow binding and so are the services, to allow SOAP header changes. Also noteworthy is the interface offered by toboggan monitor (IP, port and available operations) that allows connecting it to any Toboggan application and to the configuration files.
Figure 6.7: Toboggan Monitor Implementation Patterns
Rich Internet Applications – Study and Development

screenshots

Figure 6.8: Toboggan Monitor

List of Processing Chains

<table>
<thead>
<tr>
<th>Processing Chain Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
</tr>
</tbody>
</table>

Figure 6.9: Toboggan Processing Chains
6.3.2. Toboggan XDCAM Proxy Merger

XDCAM devices offer a low-quality version of recorded videos, called a proxy. The Toboggan XDCAM Proxy Merger is a simple desktop application that gets all the proxies from connected XDCAM devices and merges them into a single file for storage purposes.

![Figure 6.10: XDCAM Proxy Merge]

**Use Cases**

- Add / remove / Edit XDCAM locations;
- Merge proxies from selected XDCAM(s).

**Requirements**

XDCAM comes with a software that maps connected XDCAMs to virtual folders. Toboggan XDCAM Proxy checks in specific subfolders and creates a COD describing the merge operation and proxies to merge. It then sends the COD to a Toboggan controller that handles it.

**Implementation Details**

This application is adapted from the processing chains component used in Toboggan Monitor, and uses Toboggan Monitor in a tab. The configuration shown in the screenshots is done using the Submit Popup component.
Screenshots

Figure 6.11: XDCAM Proxy Merge

Figure 6.12: XDCAM Controller configuration
6.3.3. Toboggan EDLConverter

EDLConverter is the internal name of a desktop application Toboggan that reads media files from several different devices (XDCAM, P2, or a file system), converts them to a MXF file and an AAF file and forwards them into an AVID asset manager, as depicted in figure 6.13.

This application had already been developed in C# and the first task was integrating Toboggan Monitor with it. This was done using the Microsoft Web Browser C# component, and the XML configuration file described in Toboggan Monitor. To provide a better integration and interface uniformity, a Flex interface was later developed, to substitute the old one. Some of the device specific code was kept in C#, converting the functions into web services that are called from the new Flex interface.

![Sequence Diagram](image)

Figure 6.13: EDLConverter sequence diagram

Requirements

- Maintain a similar layout to the previous UI;
- Simplify the interface - Tape ID, Disk Label and Comments were not being used anymore, and the workgroup/standalone distinction can be done in the background.

Architectural considerations

Developing the new interface of EDLConverter showed the need to separate the handling of the different devices in separate services, called Toboggan Adapters. This further improved the modularity of the Toboggan applications, and suggested a new architectural approach to Toboggan products, described in the future development section (6.3.5).
Screenshots

Figure 6.14: EDL Converter old UI

Figure 6.15: EDL Converter new UI
Figure 6.16: Browsing in EDL Converter, using a submit popup component
6.3.4. Quality assurance

The Toboggan products are used by and customized for a large number of international companies in the professional media area, namely NBC for the next Olympic Games in Beijing. Apart from internal tests, the applications developed were extensively tested by the clients. The developed Toboggan products were also shown at NABShow2008 in Las Vegas, the largest broadcasting exhibition, with hundreds of thousands of attendants.

6.3.5. Future Development

Possible additional improvements on the Toboggan Monitor include:

- Editing processing chains;
- New actions over the operations (pause, postpone, cancel all failed);
- Charts and statistics on operations times;
- Optionally join the operations of several Toboggans in the same monitor;
- Improve look and effects;
- Allow actions over operations in a right click menu.

The first versions of EDLConverter and XDCAMProxyMerge only offer operations on demand, because the client requested so. Future releases can include polling of devices and locations, waiting for a new file to drop on the watched folder and automatically be processed, or scheduling of the operations dispatch.

Similarly to what was done in EDLConverter, there should be different Toboggan Adapters to handle each device, providing a standard interface, similarly to what happens with Toboggan Operations. The interfaces developed for these Toboggan applications can also be further modularized. The not so distant future of Toboggan might allow deploying any combination of web/desktop deployment, user interfaces, Toboggan Adapters and automated and on-demand Toboggan Operations running on any number of servers, all monitored in Toboggan Monitor, using a full Service Oriented and Event Driven Architecture. A diagram of this Toboggan architecture is depicted in figure 6.17.
Figure 6.17: Possible Future of Toboggan
6.4. XMLEasyConfig

Scope and Motivation

Most MOG applications offer and require a high level of configuration. It was required to build an application that would help users configure the default settings of some of MOG applications. The current configuration files had some structural similarities, so the proposed solution was to create an interface that would help navigate the XML and edit each attribute in a textbox.

Defining an XML structure with an XML file, and then specifying things like optional, fixed, default attributes and which XML nodes were reoccurring and which weren’t is the purpose of XML Schema. Therefore, I proposed a more ambitious alternative: an interface that would load a schema, and create an interface according to it. This interface would then be the intuitive way of filling or changing the configuration files, exporting the resulting XML file. A research on existing XML creation solutions brought to the conclusion that this idea is innovative, and in this age of XML ubiquity, this tool should be a very useful one.

Solution

The proposed solution involves two main graphical components that represent the XML defined in any XML Schema: The navigator and the component renderer. Figure 6.19 shows the XML navigator on the left, with a selected node “People” that is being edited on the right side rendered component. The red node (“Products”) means there are required fields yet to be edited. Any node clicked can be rendered and edited, generating the schema-compliant output XML.
In this example, the <People> XML subtree has three children, all of them without editable attributes or simple content but with children of their own. This conditions define that they are labels to its content, and can be rendered as a tab navigator. The selected tab represents the <UserInfo> XML subtree, which is a sequence of two elements, <Person> and <Relatives>. <Person> has two string attributes, each represented with a text input. <Relatives> also has two attributes, but it is a reoccurring node, which makes it renderable as a Table where rows (nodes) can be added. These rules that define components to be rendered are contained in the Pattern matching component. The XMLEasyConfig architecture, shown in Figure 6.20 and contains this component as well as the navigator (called XMLNavigator v2 because it is based on the XML Navigator reusable component) and the dynamically rendered interface. The “well defined XML” is a class that has typed attributes, mixing the XML Schema and XML functions.
Requirements

- Schemas must be implemented for existing Toboggan applications;
- All input sources in any XML must be accessible;
- The priority of features to implement was determined by the existing XML configuration files, namely:
  o `<element>`, `<complexType>`, `<sequence>`, and `<attribute>` tags;
  o type, default, fixed and use attributes;
  o enumeration restrictions.

Status and future Development

This application was interrupted by a higher priority project, the MXF SpeedRail Metadata Editor. Many important features are yet to be implemented, although the pattern matching and rendering engines are already completed. The features already implemented are depicted in Figure 6.21. There isn’t much more work necessary for the requirements to be met, but developing the ideas behind the application (dynamic generation of interfaces for xml editing) to a full extent could involve the full implementation of the XML Schema specification, deeper research on interface design patterns and possibly the development of a Schema creator from XML.
Figure 6.21: XMLEasyConfig Prototype and testing environment
6.5. **MXF SpeedRail Metadata Editor**

MXF SpeedRail is a Media Asset Manager and Ingest Station, sold as a rack mountable server with an embedded web application. It covers the ingestion and annotation phases of the profession multimedia workflow, acting as a buffer where assets are kept for annotation and then forwarded to a main storage server. The web application allows these functionalities, being intended for actors such as the ingester, the logger and the administrator of the system. The MXF SpeedRail Metadata Editor provides an interface to perform some of the ingesting and logging features of MXFSpeedRail.

**Use Cases**

![MXF SpeedRail Metadata Editor use cases](image)

An ingester goes through a video, e.g. a television emission, defining which parts of the video he wants to store. Each of these parts is a scene, which is defined by two timecodes, the marks (mark in and mark out), and the set of scenes is called a production. SDI is a digital video transmission interface supported by MXFSpeedRail. In the “append / split SDI Productions” use case in figure 6.22, a single video can be cut in many scenes that can be aggregated in different productions.
Requirements

- Modularity – The component was going to be included in different parts of the project, in many different scenarios – with or without SDI and for logging or ingesting. Specifically, it should be possible to include any combination of the yellow rectangles of figure 6.22 in an application. The layout of the components that represent each of these yellow rectangles should be changeable. The screenshots of the application show two (not very but nonetheless) different layouts;

- Connected components – Although the inclusion of a specific component in an application is unknown, there are some connections between components in case they are included, namely:
  - The scene selected on the scenes list, the marks input (video or MarkInput component) or the metadata component is the same;
  - Allowing adding or updating marks depends on the exiting scenes
  - Appending splitting productions depends on the metadata

- Maintain chronological order of the marks, and do not allow productions to overlap, i.e. a production cannot have a scene that is between scenes of another production;
- When updating marks breaks that rule, change production;
- Ask for confirmation when deleting marks or when changing productions that have already inserted metadata.
Implementation details

The uses described in the use cases rectangles are offered by the following graphical elements (top to bottom): 1st-MarksInput/Player, 2nd-SceneList, 3rd-MetadataComponent, 4th-last optional column of the ScenesList. The screenshots show these components. The player is ActiveX component that communicates with Flex through JavaScript. In the last optional column, if a scene checkbox is selected, the scene is appended to the previous. An unchecked scene is the first scene of a production. The requirements raised some complex implementation issues, which solution depended heavily on relying on the patterns described in the Flex working environment chapter and exemplified in Toboggan Monitor.

Screenshots

Create job

Action: Ingest SDI

Production: Scenes

Language: German

UMID:

Mark title: aid

Secondary title: aid

Synopsis: aid

Full description: aid

Choose Destination:
<new>

Choose File Naming:
<new>

Choose Start Schedule:
06/03/2008
14:05:40

Scenes

Mark In: 01:00:05:01
Mark Out: 01:00:06:01

01:00:05:02 01:00:06:02
02:00:00:01 03:00:00:01
07:00:00:02 07:00:00:03
08:00:00:02 08:00:00:03

Append

Update Delete

Cancel

Figure 6.24: MXF SpeedRail Metadata Editor with several productions
Quality assurance

Apart from internal tests, this application was shown at NABShow and tested by the Zurich production centre TPCAG.

Future Development

Apart from additional metadata attributes that can easily be added, the append interface design pattern could be changed to a tree component (with productions containing scenes) and the marks input removed, allowing drag-and-dropping directly from the player to a specific production and changing the order of the marked scenes if allowed.
6.6. Conclusion

The creation of reusable components and especially the implementation patterns example in Toboggan Monitor make the connection from the concepts in the previous chapters to actual implementation. The selection of software design artefacts of the numerous applications and the separation of the reusable components in this chapter hopefully will provide a good idea of the work developed in a more concise, readable and useful way than just providing the full documentation would. ActionScript E4X allowed complex XML creation and handling, which was not exposed in this document because the amount and complexity of the multiple formats, different XML conversions and professional video concepts involved are hugely simplified, hidden and summarized because they are too technical and area-specific. Development at MOG is driven by client requests and professional media exhibitions, making it very fast and with always changing objectives. The defined Flex workflow proved to be adequate for such agile environments.
7. Conclusions

The RIA technologies have a lot of potential and will play an important role in the future of software development. Serious attention from the software engineering world is required to create a strong background on building the next generation of applications. This document is a contribution in that direction.

7.1. Summary

During the RIA study, it became clear that the evolution of its technologies is motivated by the increasing need of strong languages on the client-side, to allow more complex applications without the unnecessary overload on the server. Desktop interface paradigms are mimicked in the web world and are also a motivation for the evolution of the graphics support in these technologies. The evolution has been slow and held back by the lack of browser support of complex features of scripting, vector graphics and dynamic interface languages, which motivated the appearance of proprietary plugins with sandbox environments that offer consistent browser and platform independent developing.

The RIA technologies are evolving very fast. During the 20 weeks of this project, a new version of Flex came out (version 3), two beta versions of Silverlight with colossal improvements and many toolkits and components for each of the dozen RIA technologies followed. Choosing the technology was by no means a task to be taken lightly; over one third of MOG’s developers started programming in Flex after my RIA technology study. Flex proved to be the best solution, even if not the most obvious one. Flex integrates very well in a complete enterprise development workflow and has a healthy community, detailed documentation and intelligent and mature language design, making it easy to learn and good for demanding agile environments.

As the technologies evolve and more complex applications are built on RIA technologies, there is a change in typical architectures and the application of more advanced development patterns and practices is required. An effort was made to integrate as many of the concepts learned and experience obtained during my last study years in this project, methodically analysing each step taken and searching for improvements.
7.2. Results

Experience with working in Flex supported the idea that the choice of technology and Flex study was successful and had a very positive impact. While the look of the applications could be improved, this wasn't as important as the interaction and navigation, which had good results overall. The applications passed the requirements tests, unit tests, usability tests and client tests. The applications developed had a high projection, such as the Olympic Games of Beijing, the broadcasters at NABShow and many of the leading professional media corporations.

Partially because of the Bologna process, this document tried to use a more academic, theoretical view on RIA development, instead of going too deep into implementation details. The result is this document can be used as a theoretical and practical guide to the RIA world and to Flex development.

7.3. Contribution

This project involved many options, and because of this, informed opinions were given during the technologies' description. Apart from these occasional possible subjective views on a complex environment, this project is mainly the compilation of information from many sources, the result of a first experience with many technologies and the combination of known software engineering and computer science concepts. There are many RIA technology comparisons on the web, but the most important parameters were chosen and others were added, creating a far more in-depth comparison than the usual technology comparison. The ideas exposed in the Flex working environment chapter were selected because they weren't found elsewhere, though, avoiding repeating concepts that are referred in Flex books, as well as obvious ones.

All the interface work in the development phase of this project was done alone with the exception of the EDLConverter and XDCAMProxyMerge for which I developed about 90% of the code. The back-end of the applications, although implemented at the same time than the interface, was developed by other MOG developers. The reusable components and the idea behind XMLEasyConfig are in my opinion very useful and innovative ideas that will have an impact in future development at MOG.

7.4. Future work

The next step in the study of RIA could be exploring old and new concepts of diagramming and define their adequate integration in the development workflow, with focus on automatic code generation for Flex. The introduction of the Degrafa and Openflux frameworks can be studied in the process of improving the look of the applications. Interface design patterns is a very
interesting area to study as well, and the emerging post-wimp interfaces referred in Annex A are areas prone to innovation.

The “event”, “2-way-bindable object” and “delayed instantiation” components on the “other ideas for language enhancements” section are useful additions to a component library. Apart from improving the look of the developed applications, there are several additional features described in the future work section of the description of each application that could be added and XMLEasyConfig is still not completed. Further study on the proposed architecture of Toboggan products is due and, depending on its acceptation, it might produce several changes in the architecture of both the front and back ends of the applications. When Flex evolves to include mobile development, many changes must also be introduced to the Toboggan products to support it. The already modular design of the existing code will facilitate all this changes.
References


[Gam94] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.


[MOG08] S. A. MOG Solutions. MOG Solutions - The MXF experts. Available at


D. Schall and e. al. Flex Unit. Available at http://code.google.com/p/as3flexunitlib, last accessed on Jul. 1, 2008


W. W. W. C. Unknown. XHTML 1.0. Available at http://www.w3.org/TR/xhtml1, last accessed on Jul. 1, 2008

67


Annex A - History of RIA

RIA are a result of the evolution of the web and desktop technologies. I will focus on some of the most important events in RIA’s history in a chronological manner.

1963 - Birth of the graphical user interface

Xerox 8010 Star UI

- Concepts introduced, like WIMP UIs—windows, icons, menus and pointing devices— or WYSIWYG (What You See Is What You Get), created a background for most of today’s interfaces. In figure A.1., it is possible to see how similar the first graphical interface is to current interfaces.

![Figure A.0.1: Xerox 8010 Star UI](image-url)
1991 – *Birth of the WWW*

![First WWW browser](image)

1992 – *First wide-spread browser – Mosaic*

1994 – *Netscape Navigator, later to become open source Mozilla and spin-off Firefox*

1995 – *Microsoft releases Internet Explorer*

**Browser Wars**

- The fight between Netscape Navigator and Internet Explorer had significant consequences. Each browser wanted to have more features than the other, but fixing bugs was not as important. It also led to the adoption of non-standard features like style attributes in XHTML and the creation of proprietary features. Figure A.3. shows the rise of Internet Explorer, with its dominance marking the stagnation years described further in this annex.
• In 1995, Netscape introduced JavaScript, which allowed pages to be dynamic for the first time (DHTML), i.e. change without reloading. JavaScript became widely used, even with its many flaws in language design and in security.

• Macromedia introduced shockwave and Sun introduced java applets, marking the beginning of more complex software embedded on a webpage.

• The web business started to grow incredibly fast and the speculation around it would lead to the dot-com bubble.

1996 - Macromedia Flash, IE3 and Swing

• A year later has passed since Microsoft introduced Internet Explorer and it is already on its third version. IE3 supports ActiveX, java applets and some CSS1 features (although only in 1999 did it offer CSS1 full support and it is still to offer CSS2 support).

• Macromedia introduces Flash.

• Java Swing makes Java’s (and applets’) interfaces easier and more appealing.

1998 - Remote Scripting

• Motivated by a new Microsoft Outlook functionality, Internet Explorer introduced Remote Scripting, later to stimulate AJAX.

1999 Macromedia introduces ActionScript
• In spite of being a very simple language in initial versions, ActionScript allowed a greater complexity in Flash assets. There was a mass acceptance of Flash by designers and a proliferation of flash websites and a lot of ads, creating a lot of lovers and haters.

2001 – *Web stagnation, Mac OS-X Aqua interface*

• The browser wars ended, IE has the monopoly and there won’t be many new features or bug fixes in the next 5 years.

• The dot-com bubble bursts and many web companies go bankrupt.

• The Mac OS-X interface shown in figure A.4 had a deep impact on interface designers. It marks the beginning of widely spread use of concepts and effects like fish-eye menus, shadows, transparencies, reflection and compositing.

![Figure A.4: Mac OS-X UI](image)

2003 – *XForms*

• The ideas behind the XForms specification influenced other XML-based interface technologies (Flex, Silverlight, ...)

2004- *Flex & Web 2.0*

• The release of Flex started to attract programmers to the previously designer-oriented world of Flash.

Web 2.0 hype

• Tim O’Reilly introduced the buzzword Web 2.0, which described the rising trends in web development, starting a new hype around web development. Figure A.5. exhibits the main concepts, motivations and propellers of Web 2.0.

• Ajax gained notoriety and a new period of growth in web business started.
2007 – The RIA year

- Vista and Ubuntu incorporated the popular Mac OS-X interface ideas.
- Microsoft released Silverlight (as a subset of WPF), which marked their commitment to the XML-based languages.
- JavaFX, Java's RIA technology, was announced.
- Adobe released Air, the platform that provides desktop deployment for AJAX, Flex and ActionScript.
- Adobe open sources Flex SDK and cooperates with Mozilla on ECMAScript4 (the standard on which ActionScript follows and JavaScript is based on) compliance for Firefox4.
Because of Firefox's slow but constant market share growth and the new wave of interest in web business and RIA, Microsoft is again improving IE, this time following standards a bit more (although some fear that the introduction of Silverlight might reduce the chance of IE integrating SVG, XForms or other open standards that compete with Silverlight).

Apple introduced the iPhone, with some uncommon interface concepts, like multi-touch and Zoomable User Interfaces.

Future of RIA

This section is obviously full of speculation, but nonetheless an informed type of speculation. It is very important to predict where the technologies are heading when developing in an ever-changing environment like RIA.

Deploy everywhere

The next huge step is to be able to deploy the same application not only in all operating systems, but also in all platforms. An application developed only once will in handheld and embedded devices, through the net, or on any desktop, in a seamlessly integrated cross-device environment.

Development

Adobe will release Thermo, a very powerful and imaginative designer tool that can convert images into Flex interface components and add logic to it automatically. Flex 4 will introduce the rest of Flash's graphical power to Flex by splitting each of the existing interface components in a Model-View-Controller of its own, making them completely configurable and reusable. These enhancements to the Flex platform might bring more of the highly populated world of designers the use Flash to Flex.

Project Tamarin is the result of the contribution of Adobe's ECMAScript4 virtual machine to Mozilla. This will allow Firefox to run JavaScript 2, but it also means that a lot of ActionScript's functionality will be possible to run on the browser. The implications of this in Adobe's strategy and Flash player's code are not clear. Flash Lite is getting a lot of attention and will probably get integrated in much more mobile platforms. Tamarin Tracing is the low CPU, low memory version of Tamarin and it will probably be the next ES4/AS3/JS2 virtual machine for the mobile world.

Silverlight will run on Windows Mobile. Since Silverlight is a subset of Windows Presentation Foundation and thus further features already included in desktop interface development might cross the border to the web and mobile platforms. This uniformity in .NET interface development might stimulate desktop developers to switch to RiA deployment, and the
.NET framework is enormous and includes several programming languages (Jscript, VB, C#, Python).

JavaFX is in its infancy, and with time we will see more and more RIA being developed with it and deployed on the desktop, web and mobile worlds, where Java is already huge. Also new in Java is Nimbus the new look and feel that is a lot more appealing than the old one. Java webstart/applets will have major improvements regarding installation time, loading time and API management.

**Lots of RIA**

Because RIA technologies’ documentation getting a good attention and the high-level features, it is getting easier and easier to develop RIA. The Web 2.0 buzz will continue and the same way the dot-com bubble saw many people, programmers or not, creating XHTML pages for their personal interest, RIA will continue to appear exponentially.

**Post-WIMP interfaces**

3D interfaces are becoming common in data-heavy navigation problems (e.g. music recommenders) and used in compositing window managers, widgets and many effects. Technologies like Papervision3D (ActionScript 3D library) are raising attention, and they will be incorporated in some RIA graphic interface components. Full 3D interfaces have been experimented, but, apart from the virtual worlds’ field, they show flaws like hidden objects and orientation problems. Zoomable User Interfaces are a strong candidate for a next shift in interfaces, since they avoid those 3D drawbacks. There are numerous applications for these interfaces, and some technologies that potentiate their use. Microsoft Photosynth and Deepfish, for example, are very powerful technologies that might trigger ZUI.

On the handheld and embedded devices world, the multi-touch interfaces will also be important in shifting interface paradigms. Microsoft Surface, Perceptive Pixel and the iPhone show the first practical uses in the embedded, desktop and hand-held environments.

**Open standards**

The evolution of JavaScript to the current ECMAScript standard would provide us a very complete scripting language. It might happen in the near future, as Microsoft’s Silverlight emphasises their JavaScript implementation (JScript) and Firefox4 will incorporate ECMAScript4 (with Adobe’s help). The open world of AJAX is very alive, with frequent library-based improvements.

The evolution of XHTML to the very dynamic and powerful XHTML 2.0 would bring the most powerful XML-based layouts, although the less interesting HTML5 (adding features to HTML rather than redesigning it) is more likely to be the next step. On the presentation side, CSS adoption seems to be in browsers priorities, but not SVG (at least for IE).