



ÉCOLE
POLYTECHNIQUE
DE LOUVAIN

Université Catholique de Louvain
MSc Thesis

*Model-Driven Engineering of User
Interfaces for Rich Internet
Applications: the case of the Oliva
Nova Model Execution.*

by

Quentin ENGLEBERT

supervisor

Jean VANDERDONCKT

August 22, 2009

Contents

1	Introduction	1
1.1	Concepts	1
1.1.1	Web development	1
1.1.2	Rich Internet Applications	2
1.1.3	Computer-Aided Software Engineering	5
1.2	Goals of the MSc Thesis	6
1.2.1	Model-driven engineering	6
1.2.2	Beautification	6
1.2.3	Thesis statement	7
1.3	Hypothesis	7
1.4	Methodology	8
1.4.1	Chosen CASE tool	8
1.4.2	Chosen programming environment	8
2	State of the Art	9
2.1	Features of RIA applications	9
2.2	Programming environments	12
2.2.1	Ajax	12
2.2.2	Adobe Flex	16
2.2.3	Microsoft Silverlight	17
2.2.4	OpenLaszlo	18
2.2.5	Google Web Toolkit	18
2.2.6	JavaFX	19
2.2.7	Echo3	19
2.2.8	haXe	20
2.2.9	Curl	20
2.2.10	FlashiXML	21
3	Technological choices for RIA production	22
3.1	ONME	22
3.1.1	MDA and the OO-Method	22
3.1.2	Presentation Model	23
3.1.3	Oliva Nova Model Execution	23
3.2	Ajax	24
3.3	JQuery	24
4	Design patterns for RIA	26
4.1	ONME patterns	26
4.1.1	Basic Elements	26
4.1.2	Interaction Units	27
4.2	Results of the analysis of RIA constructs for ONME	27

5	Implementation of significant RIA constructs	30
5.1	Master-detail with a slider and an accordion	30
5.2	Carousel for navigation basic element	31
5.3	Imagecropper for Available Actions basic element	33
6	Development of a software to support RIA production	35
6.1	Software approach	35
6.1.1	Beautification process	35
6.1.2	Beautification editor for ONME	38
6.2	Specification of the tool	38
6.3	Implemented features	39
6.4	Implementation Architecture	43
6.4.1	Packages	43
6.4.2	Classes	43
6.4.3	Adding a widget	51
6.4.4	Adding a pattern	52
6.4.5	Implementation effort	52
7	Conclusion	53
7.1	Contributions	53
7.2	Improvements	54
7.3	Final word	54

Introduction

In the middle of the 1990s, the explosive growth of the Internet lead to the general adoption of a model for applications with low developing costs while increasing application types that could be offered. This model was based on a very thin client and huge servers that created dynamically web pages before delivering them to the clients.

This model recently began to show its limits because of the increasing complexity of tasks performed through web applications. Rich Internet Applications (RIA) have become more and more important those last years thanks to: [NODA05]

- broadband increase
- computing power shift between PCs and servers
- a need for sophisticated user interfaces
- a move from leading tech companies
- the emergence of Web Services and Service Oriented Architecture (SOA)

However, the RIA market is still immature. Technologies are evolving every day and there is a lack of specific methodologies to design Rich User Interfaces.

Some work has been done these last years to develop a proper model related to Model-Driven Architecture (MDA) to reduce developing costs and produce flexible interfaces [MA07]

1.1 Concepts

In this section, we will introduce the main concepts appearing in this MSc Thesis. We will first explore the domain of web programming to have a small overview of today's technological choices. Then we will define RIA, see their advantages and current design issues to understand where efforts must specifically be done. Afterwards, we will have an overview of CASE tools to understand how they can address current design issues of RIA.

1.1.1 Web development

HTML (Hypertext Mark-up Language) allows to describe the structure of web pages: it lets web developers format text, add images, create links, forms, tables, . . .

While HTML might be sufficient for simple document-like web pages, typical web development consists of more areas, among others server-side coding, database technology and client-side coding.

- **Server-side coding** Some operations are performed server-side because they require access to information or functionality that is not available on the client or because it would be unreliable client-side. They also allow to process and store data from the client to the server. In particular, **server-side scripting** (such as PHP) allows running a script directly on the server to generate dynamic web pages. It is often used for interfaces to databases. The advantage of server-side scripting is being able to customize the response according to user's preferences, access rights or queries.

- **Database technology** Databases are used to store data on servers to be accessed anywhere (for instance electronic mails) or for coordination between many users.
- **Client-side coding** Some operations are performed client-side because they require access to information or functionality that is available on the client but not on the server, because they require interaction with the user or because the server lacks the processing power to perform the operations in a timely manner for all of the clients it serves.
In particular, **client-side scripting** (such as Javascript) allows running a script on the client (for instance in a web browser). They require that the client understands the scripting language in which they are written. Their main advantage is that processing can be done without sending data over the network, they take less time than server-side scripts, use less bandwidth, and are more secured for the client (while they are not allowed to access the user's computer).

1.1.2 Rich Internet Applications

We will now define precisely the concept of Rich Internet Applications (RIA) and discuss current design issues.

Definitions from the literature

Various definitions of RIA exist in the scientific literature and on the Internet, and as we will see, the concept is ambiguous.

Let's have a look at the first functional definition of a RIA. In March 2002, in a document describing the technical aspects of *Macromedia Flash MX*, the crucial aspects of a Rich Client Technologies were described for the first time [[ALLA02](#)]. Rich Client Technologies should:

- Provide an efficient, high-performance runtime for executing code, content and communications.
- Integrate content, communications, and application interfaces into a common environment.
- Provide powerful and extensible object models for interactivity.
- Enable rapid application development through components and re-use.
- Enable the use of web and data services provided by application servers.
- Embrace connected and disconnected clients.
- Enable easy deployment on multiple platforms and devices.

Today, we can find the following definition on Wikipedia [[WPRIA](#)]:

Rich Internet applications (RIA) are web applications that have some of the characteristics of desktop applications, typically delivered by way of a proprietary web browser plug-ins or independently via sandboxes or virtual machines.

Here is a more precise definition [[MART06](#)]:

RIA are Web applications that transfer most of the load of processing the user interface to the Web client while the predominant part of data (from control and maintaining to business data) remains on the application server.

The following definition of RIA is more general [[NODA05](#)]:

Rich Internet Applications (RIA) capitalize on the strengths of both web and desktop applications. RIA is a set of technologies which enables some or all of the characteristics [of those applications].

Here is another definition based on their advantages [BOZZ06]:

They provide sophisticated interfaces for representing complex processes and data, while minimizing client-server data transfers and moving the interaction and presentation layers from the server to the client.

A definition based on their features also exists [PREC07]:

RIA offer online and offline capabilities, sophisticated user interfaces, the possibility to store and process data directly on the client side; they offer high levels of user interaction, usability and personalisation, minimise bandwidth usage, and separate presentation and content at the client side.

Comparison of definitions

While giving the advantages or features of such applications might be useful to convince developers to use them, it isn't suited for a theoretical definition of RIA. The definition should describe what a RIA is, and not how it works or what it does.

There is a common belief that RIA consist in adding AJAX functionality to existing websites. But can a website like Youtube (<http://www.youtube.com>) be considered as a RIA? This website uses Flash to display content but isn't really an application. So we have to emphasize that a RIA is a **Web Application**, to avoid such confusions.

After having seen every definition, we should agree that **RIA take advantage of both web and desktop applications**: now that Web Applications have become more complex and popular, processing can't only be done server-side anymore and RIA are designed to cope with this problem by processing data client-side. Here are the advantages of desktop applications [NODA05]:

- Richer user experiences
- No page reloading
- Online and offline support
- Can be more complex
- More responsive and interactive

Some definitions are already out-of-date because of fast evolution of RIA:

Can we still say today that RIA only process the User Interface? The Internet is in constant evolution, and today, RIA are even replacing the old traditional desktop applications. An example is Google Documents <http://docs.google.com/>, now even available offline!

Indeed, using RIA instead of desktop applications benefit to both customers and providers [LASZ05]:

- Application providers can deliver applications by servers, which reduces costs.

- Users don't have to configure nor update applications and can access services everywhere.

RIA can even be used outside a Web Browser, for instance in a Site Specific Browser like Prism (<https://wiki.mozilla.org/Prism>).

Proposed definition

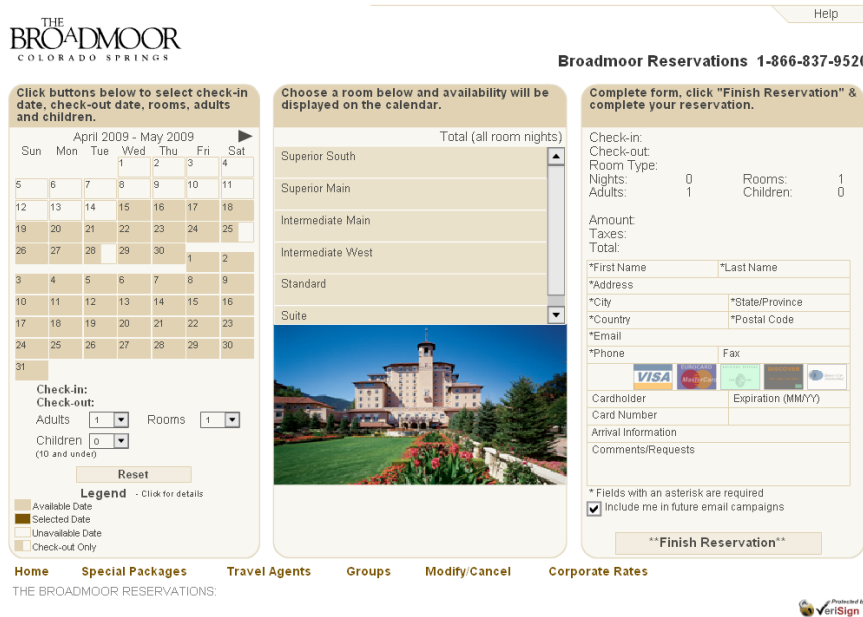
Current aspects as the environments using RIA (browsers, sandboxes, virtual machines), the amount of processing done client-side (what is currently executed client-side and what remains on the server), their advantages and what they offer in the current architecture of the World Wide Web should not be given in a definition to avoid they become outdated it in a few years after more evolution. Therefore, to deal with the constant evolution of the domain, we will stick to a general definition based on their main goal:

RIA are Web applications that capitalize on the strengths of both web and desktop applications: they transfer the processing of various elements (such as the User Interface) to the client.

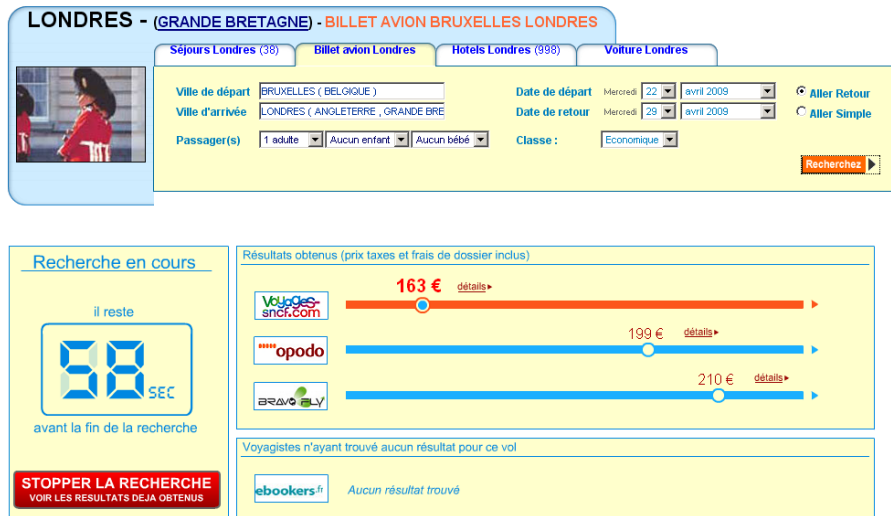
Examples

Here are some examples of RIA:

- **Broadmoor Hotel** (<http://www.broadmoor.com/>): The reservation process on this website is much less frustrating than standard processes where the user is forced to page back and forth when the price or the date doesn't match his expectations. The whole reservation process can be done on one page. Every task is directly executed without pages being reloaded.



- **Alibabuy** (<http://www.alibabuy.com/>): This website gives a price comparison between flights from various companies. The search takes 60 seconds but the tool seeks in realtime so the user can see the first results while others are processed. Then the user can change search fields and make a new search without any page loading.



Current problems

As the technology is still in its evolution, each class of contributors has currently some issues to face:

Designers faced issues with conceptual modeling for RIA in data modeling, hypertext modeling in the large and hypertext modeling in the small [BOZZ06] so there is currently a lack of specific methodologies to develop RIA, such as model-based design.

Developpers have to become familiar with developping tools. Learning a framework can take some time, and there are multiple RIA platforms, each one having its own advantages over others [NODA05].

Users of web applications belong to a wide number of categories from novice to experts. They have various requirements for user interfaces. Too many new gadgets can also cause cognitive overload to them [MART06]. They can also have bandwidth issues to download RIA.

1.1.3 Computer-Aided Software Engineering

Computer-Aided Software Engineering (CASE) is the use of computer-based support in the software development process. It helps to provide high-quality, defect-free, and maintainable software products.

A **CASE tool** is a computer-based product aimed at supporting one or more software engineering activities within a software development process [SEI07].

CASE tools have two main kinds of utility (some integrate one of them, some integrate both):

- Providing computer assistance in software development (and or maintenance) processes, as model transformation tools do.

- Providing an engineering approach to software development (and or maintenance), as UML designing tools do.

There is a need to connect CASE tools together (for instance to have a single set of documentation with a centralized version control), so a mechanism is needed to support interactions among CASE components. So, a **CASE environment** is a collection of CASE tools and other components together with an integration approach that supports most or all of the interactions that occur among the environment components, and between the users of the environment and the environment itself [SEI07].

1.2 Goals of the MSc Thesis

1.2.1 Model-driven engineering

Model-Driven Engineering (MDE) is a software development methodology aiming on creating and transforming models. An example is the well-known Unified Modeling Language (UML) [WPMDE].

MDE allows model-to-model (M2M) and model-to-code (M2C) transformations, which can be done automatically by CASE tools.

The need for a model-based design for RIA has been growing those last years because of the increasing number of applications to be build and conditions that applications should fulfill [MART06].

1.2.2 Beautification

Some user requirements aren't supported by M2M and M2C transformations.

The beautification consists in manual changes to deal with user requirements that have not been supported during transformations. The need for these changes comes from user's preference, customization or compliance with corporate style guidelines [P&a107]

Manual modifications can lead to several issues [P&a107]:

- Problems to understand the generated code
- Inconsistencies between the final application and its model
- Quality features guaranteed by MDE may become invalidated (for instance, manual changes can produce errors)
- When the model of a manually modified application changes, the new application might become no longer compliant with previous manual changes

The last issue is called round-trip engineering. There are solutions to deal with this problem: manual modifications can be saved, interpreted, abstracted then replaced by a beautified operation for the model that initiated the M2M and M2C transformations so that the operation can be replicated when transformations are reapplied [P&a107].

The first three issues occur when manual modifications are poorly designed. Therefore, avoiding manual modifications if possible is the right choice to ensure quality applications. This solution

becomes available when all necessary beautification operations are implemented to be done automatically according to user's choice.

To summarize, the conception of a framework allowing the user to choose operations to apply among pre-made beautification operations would be efficient to deal with unsupported requirements.

1.2.3 Thesis statement

After discussing the motivations of this MSc thesis, we can now state what will be defended in this paper:

The conception of a framework to allow automatic *beautification operations* improving automatically generated *User Interfaces for Rich Internet Applications* while preserving the quality features guaranteed by *Model-Driven Engineering*.

1.3 Hypothesis

In this section, we will enumerate the assumptions that have been done for this MSc thesis and explain why they were necessary.

- **Focus on UI beautification** 64% of the beautification modifications to deal with unsupported user requirements are related to User Interfaces (UI) [P&a107]. Therefore, we will focus on UI in this MSc thesis to deal with the largest source of issues in a first time.
- **Focus on designing a prototype tool** The tool developed for this thesis will act as a prototype. There is a huge number of beautification parameters that could be used depending on various needs. The complexity of the automation of some beautification operations can also be important and may require important developing efforts. The goal of this thesis is mainly to show the applicability of such a tool.
- **Forbid any manual intervention** As seen when beautification was defined (subsection 1.2.2), manual beautification operations should be avoided when possible. For a public release, the need for specific categories of users (with knowledge of the field) to add their own operations manually may arise. In the context of the prototype application (ie that isn't expected to be distributed), all required beautification operations can be implemented to be automatic.
- **Focus on analysing existing constructs** Another goal of this thesis is the analysis of the applicability of RIA constructs to patterns of the chosen CASE tool. Therefore, we will focus only on existing constructs.
- **Use of existing scripts** While knowledge of web development scripting languages is necessary for the prototype, a lot of open source scripts are available for RIA constructs. Developing new scripts for RIA constructs is out of the scope of this thesis.
- **Focus on client user interfaces: no databases** We already made the assumption that we will focus on UI modifications in this thesis. The scripts in the implementations of chapter 5 and the prototype are client-side only: we won't care about client-server interactions and databases.

1.4 Methodology

1.4.1 Chosen CASE tool

OlivaNova tool [CARE09] allows to represent user interactions with a pattern-based approach (for a more detailed overview of OlivaNova, see section 3.1.3 on page 23). It takes class models, functional models, and presentation models and creates a completely functional and executable software application.

Furthermore, it has been proven that the usability issues of an interactive application generated with the OlivaNova tool can be addressed as an integrated part of the system design and not only as an ad-hoc solution after completing most of the development [ABRA08].

1.4.2 Chosen programming environment

The prototype is developed with Java on Eclipse. RIA widgets are developed with Javascript embedded in HTML on Aptana platform, using several libraries such as JQuery [JQ], JQueryUI [JQUI] or YUI [YUI].

State of the Art

The first part of this chapter explores deeper the concept of RIA by covering their domains of applications with some examples. The second part presents the various programming environments for RIA and compares them.

2.1 Features of RIA applications

Following is an overview of some features and widgets provided by RIA and examples of websites where they may be used:

- **Auto-completion (fig 2.1):** Used in search engines to give suggestions



Figure 2.1: Google Suggest feature (<http://www.google.com/>)

- **Dashboard, drag&drop, tabs (fig 2.2):** Used in content syndication to display information from various sources

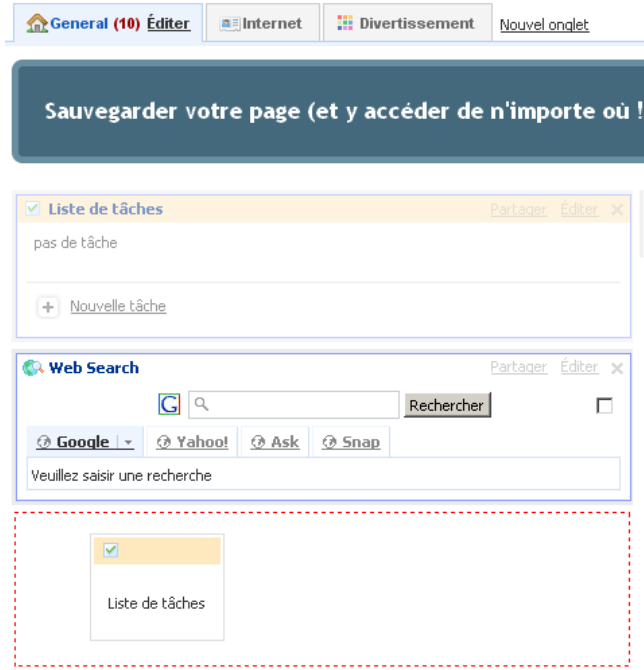


Figure 2.2: Netvibes (<http://www.netvibes.com/#General>)

- **Filters (fig 2.3):** Used in filter engines, allow dynamic presentation of results

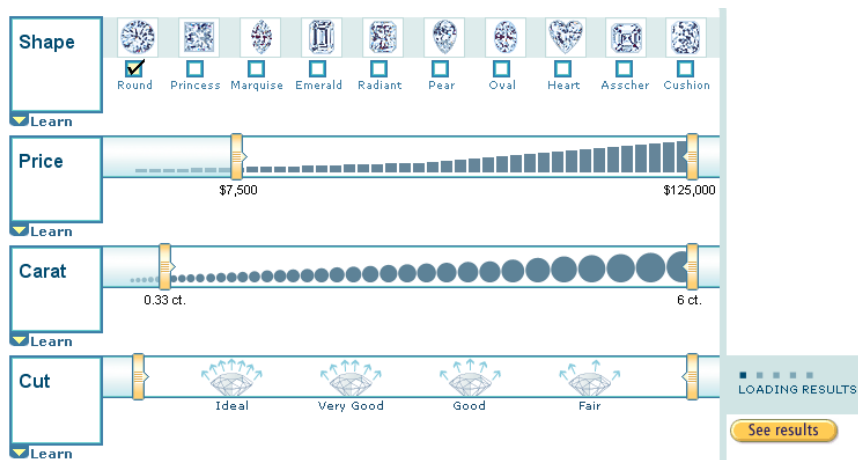


Figure 2.3: Amazon Diamond search (<http://www.amazon.com/gp/gsl/search>)

- **Keyboard shortcuts (fig 2.4):** Used in webmails

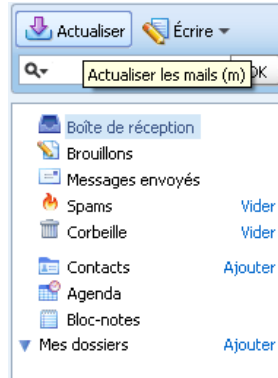


Figure 2.4: Yahoo! mail (<http://mail.yahoo.com>)

- **Document edition (fig 2.5):** Used in online word processors, allows them to be collaborative

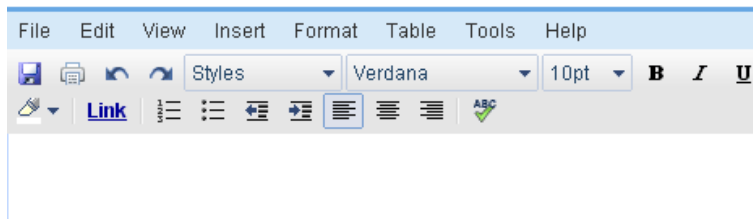


Figure 2.5: Google Documents (<http://docs.google.com/>)

- **Real-time data validation (fig 2.6):** Used in forms to check data before the user submits them

Signup

First Name ✘ First name is required

Last Name ✘ Last name is required

Username ✘ Minimum is 2 characters

Password ✔

Confirm Password ✘ Passwords do not match

Email Address ✘ Invalid email address

Figure 2.6: Remember The Milk (<http://www.rememberthemilk.com/signup/>)

- **Navigation tree (fig 2.7):** Used to navigate through large collections

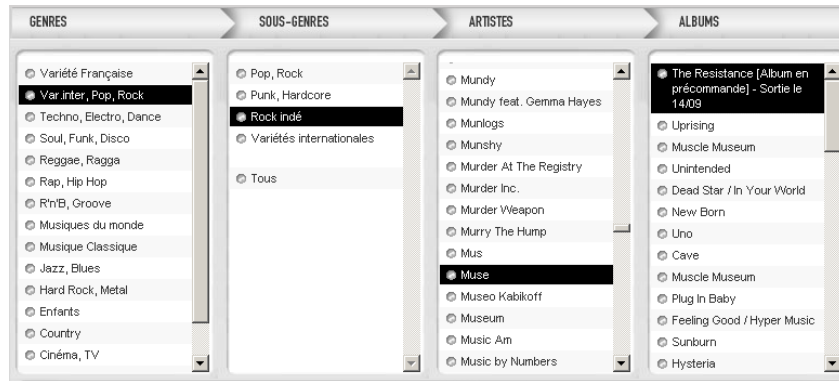


Figure 2.7: Virgin Music Explorer (<http://www.virginmega.fr/musique/explorer.htm>)

2.2 Programming environments

Several technologies are available to support RIA development. They can be classified into four categories:

- **Scripting-based:** The client side logic is implemented via scripting languages and interfaces are based on a combination of HTML and CSS
- **Plugin-based:** Rendering and event processing are done by browser plug-ins interpreting XML, programs or media files such as Flash
- **Browser-based:** Rich interaction is natively supported by some browsers that interpret declarative interface definition languages (such as XUL - the XML User Interface Language - for Mozilla)
- **Web-based desktop technologies:** Applications are downloaded from the Web but executed outside the browser (Java Web Start, Window Smart Client, Adobe AIR).

Here is a presentation of today's programming environments with their advantages and weaknesses:

2.2.1 Ajax

Description

Ajax (an acronym for Asynchronous JavaScript and XML) is not a technology: it is a combination of web development techniques used on the client-side. It was invented in 2004. It usually combines:

- XHTML and CSS for standards-based presentation (respect of W3C recommendations)
- the Document Object Model (DOM) for dynamic display and interaction, XML and XSLT for data exchange, storage and manipulation
- the XMLHttpRequest object for asynchronous data exchange with the web server
- Javascript to bind everything together and act on the classical web presentation layer thanks to the DOM

Actually, despite the name, the use of JavaScript and XML is not actually required, and requests don't even need to be asynchronous:

- Other scripting languages than Javascript, such as VBScript can implement an Ajax application
- There are many other ways to interchange data than XML, such as the Javascript Object Notation (JSON), preformatted HTML or plain text

Figure 2.8 explains the interactions of these techniques.

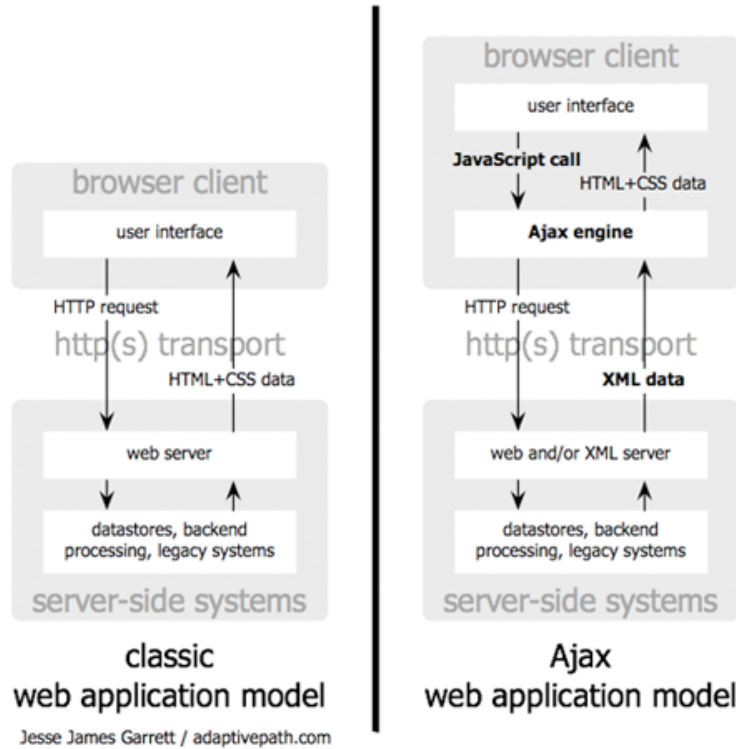


Figure 2.8: Comparison between classic web application model and Ajax web application model [GA05]

The base principle is intercepting events occurring on the page with Javascript and dynamically inserting content from a web server (usually exchanged with an XML document), with Javascript too. Those user actions are done with an XMLHttpRequest object (XHR) allowing Javascript to process a request to the server that will be invisible to the user, won't have to reload the entire page and will be asynchronous (allowing the user to do something else while the information is processed by the server).

Figure 2.9 shows how synchronous web applications require a trip back to the server for each data transmission, limiting user's activity whereas asynchronous web applications don't stall user's interactions with the application.

Strengths

Here are the advantages of asynchronous web development (common to all asynchronous techniques):

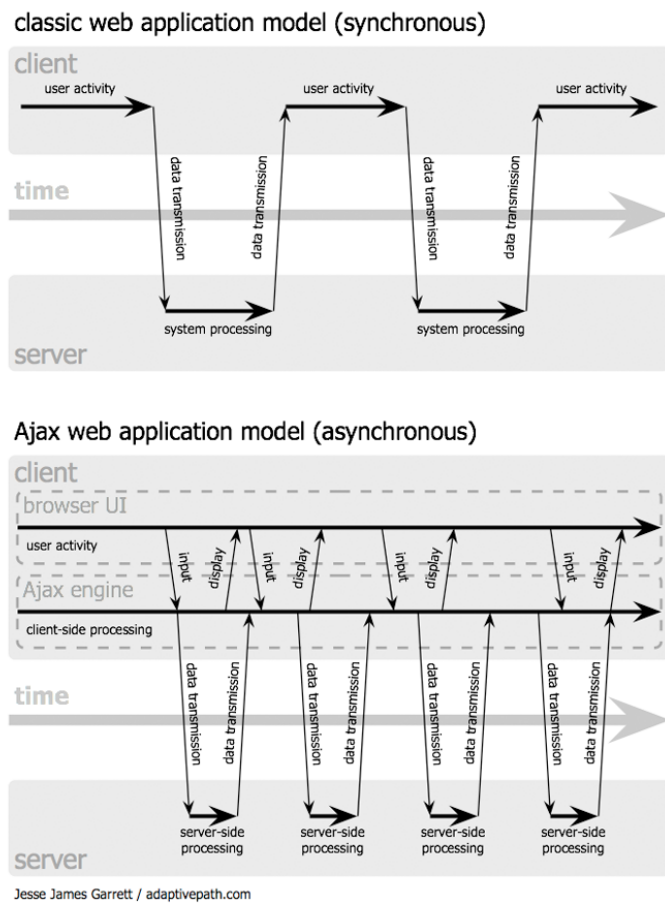


Figure 2.9: Comparison between the synchronous interaction pattern of a traditional web application and the asynchronous pattern of an Ajax application [GA05]

- **Better performances for the client and reactivity:** No complete refreshing of the web page. The user doesn't have to wait. The client's Web browser UI will respond quickly to inputs
- **Better performances for the server:** Databases queries are done only when necessary, not each time the page is requested. Scripts and style sheets only have to be requested once (only dynamic data have to be reloaded)
- **Concurrent actions:** Multiple requests are processed in parallel

These advantages have led to an increase in interactive animation on web pages and to better quality of Web services.

Following are the advantages related specifically to Ajax:

- **Powerful:** It solves current business problems: Ajax technologies allow users to interact with data in a flexible way similar to the desktop
- **Platform independence:** Developers may choose among many technology providers, commercial and open source to find the matching products or open source technologies
- **Skill-set Conformity:** A lot of developers already know HTML and JavaScript. Developers don't have to jump into new technologies of narrow use
- **Network Effects:** Ajax is widely used. As more and more people use a particular language, the value derived from using it increases exponentially. For instance, as more people start using Ajax, more resources, information, and 3rd party components, toolkits, and libraries become available. Other platforms (mobile devices, other presentation frameworks, ...) will adapt to comply with the Ajax architecture, and even begin to support it
- **Browser and platform compatibility:** Browsers are transparently compatible and Ajax doesn't require any plug-in (uses the native language)
- **Compatibility with existing technologies:** Ajax is fully compatible with the current HTML application development infrastructure: application servers (J2EE, .NET, ...), server scripting languages (ASP, JSP, PHP, ...), server application frameworks (JSF, Struts, ...), web services and service oriented architecture (SOA)
- **Multiple alternative Ajax programming models:** Ajax offers a wide range of architectural options

Weaknesses

Here are the weaknesses and challenges related to Ajax:

- **Development complexity:** Server-side developers must understand how presentation is done in HTML client pages and how to generate the XML content for the client HTML pages. HTML page developers must learn Javascript. Sophisticated development is hard (Javascript code is tedious to maintain and can easily become unstable), but components may help
- **Accessibility:** Not all browsers have access to Javascript and some users choose to disable it. An alternative solution has to be implemented for those users

- **Browsers differences:** Accessing and using the XMLHttpRequest object in Internet Explorer is different than in other browsers. Both cases must be treated separately
- **Ergonomics:** Because pages dynamically created with Ajax do not automatically register themselves with the browser's history, so some standard features may malfunction (solutions exist though, such as implementing a way to save the current page state or use recent frameworks):
 - Bookmarking or linking to a particular state of the application isn't possible by default
 - Saving pages or indexing their content isn't possible by default
 - The "Back" button may return to the last full page visited before it, not to the state before the Ajax update
- **User's feedback:** Not informing the user of pages content update or latencies due to long requests to the server may be misinterpreted by users: they may think the application is not working as expected
- **Security flaws in Javascript by design:** Javascripts can do anything the user can do: a script can pretend the user clicked on a particular link, or typed in some text
- **XMLHttpRequest object:** Due to security constraints, only information from the host that served the initial page can be accessed. Displaying information from another server is not possible according to AJAX

2.2.2 Adobe Flex

Description

Flex provides developers with a framework of extendable classes and an Eclipse-based IDE (Flex Builder) which allow them to create RIA using the ActionScript and MXML programming languages. The MXML and ActionScript in a Flex application are translated into ActionScript then compiled into a SWF file that can be read with Flash Player. Users can request these applications from a server. These applications can dynamically request data. The request goes through the application server to the database, back to the application server then back to the browser. Rich Application interfaces can be tied into existing services by making remote calls using the RemoteObject tag.

Strengths

- **Browser and platform compatibility:** Flash Player has the ability to display Rich User Interfaces consistently across platforms and browsers
- **Increasingly powerful development tools:** Flex Builder offers an easy to use development environment with a really handfull interface and features such as Design view and a debugger. It is built on Eclipse, so it is possible to have Flex code and application server code in a single IDE. Flex Builder also includes design mode of Flash for non-programmers interested in building Flex apps
- **Multimedia:** Flex (using Flash runtime) has more access to the multimedia components of computers (for instance to embed video in web pages, such as in Google Video video.google.com)

- **Speed and flexibility:** As Flex files are compiled, the execution is much faster for rich interaction and media heavy pages (there is no need to reload files client-side). Flex also efficiently uses caching for improved speed ([FLCA]).
- **Open source and alternative solutions:** Flex SDK is open source and there are open source solutions as well such as Flashdevelop (<http://www.flashdevelop.org/community/>). There are also alternative commercial options such as IntelliJ (<http://www.jetbrains.com/idea/>)

Weaknesses

- **Lack of trust from enterprise software developers:** The Flash format is a published specification (allowing alternative tools to make Flash files). However unlike open initiatives the future of the Flash platform is controlled by a single organization (Adobe)
- **Skill-set required:** ActionScript and MXML are similar to JavaScript and XML but are not as common. There are less networks effects than in Ajax
- **Limitations of Flash Player:** There are a few limitations due to the use of the Flash Player:
 - Because Flex applications are thin clients, there is a load when running a flex application initially that may last a few seconds so Flex applications are not suited for web pages but can be used similarly to desktop applications (one-time loading time required)
 - The Flash Player is not a web browser so it is harder to produce rich documentation than with HTML.
 - Flex windows must be displayed within the dimensions of the instance of the Flash Player that created it
 - Back button and right-click don't have desired effects without enhanced coding
- **Penetration:** Flash player is installed on 99% of desktop computers in mature markets [FLPE], but it is not supported by all mobile devices (for instance the iPhone) and is less present in emerging markets. Though it is more present than Java on browsers [STAT].

To conclude, both technologies have their advantages, flex being more suited for multimedia content and Ajax for large amount of text (better handled by the browser).

Both technologies can also work together. Adobe has created a Flex-Ajax Bridge facilitating communication between JavaScript and Flex SWFs. It is also possible to create widgets or sections of a website with Flex and other sections with Ajax because Flex SWF can be added to any web page.

2.2.3 Microsoft Silverlight

Description

Silverlight is a direct competitor of Adobe Flex. It is based on a subset of XAML (eXtensible Application Markup Language) to allow developing cross-browser (supports all browsers), cross-platform and cross-device applications: there are runtimes for each platform to allow the execution of applications on all browsers. The XAML code can be combined with Javascript code to offer a real interaction between the application and users. Silverlight allows to use ASP, .NET, Ajax and Microsoft tools (Expression, Visual Studio ...).

Strengths

- **Skill-set Conformity:** There is no need to learn a new language such as ActionScript: development can be done by using only XAML and Javascript. It is possible to program with multiple languages such as C#, C++, javascript and visual basic. Existing Microsoft tools such as Visual Studio 2005 and Expression Studio can be used.
- **Runtime:** The Runtime to allow the execution of Silverlight applications is light (about 1MB) and cross-browser. It is now offered with Windows Update

Weaknesses

- **Penetration:** Silverlight is still in an early stage, and the runtime is less present than Flash Player. It is installed on less than 1/3 of browsers [[STAT](#)]
- **Few open source support:** The Linux version has been delegated to Novell in the framework of the Mono Moonlight project and doesn't support the latest developments.

2.2.4 OpenLaszlo

Description

OpenLaszlo is an open source platform consisting of the LZX programming language and the OpenLaszlo Server:

- The source code is written in the LZX language which is XML with embedded JavaScript code
- The OpenLaszlo Server provides a compiler that translates LZX applications into binary SWF files that the Flash Player can execute directly or DHTML files

Strengths

- **OpenSource:** Free and customizable by anyone
- **Flexible:** OpenLaszlo is ideal to run the same code on a lot of different devices and platforms, such as Flash and DHTML (currently supported), and other platforms in the future such as Java ME, iPhone or Silverlight
- **Skill-set Conformity:** There is no need to learn a new language such as ActionScript. The development can be done only with XML and Javascript

Weaknesses

- **Popularity:** There are no big advertisement campaigns, it is less known than Flex, so there are few third party plug-ins and modules or less advanced existing components

2.2.5 Google Web Toolkit

Description

Google Web Toolkit (GWT) is an open source set of tools allowing web developers to create Ajax applications in Java.

Strengths

- **Open-Source:** GWT is licensed under the Apache License version 2.0
- **Skill-set Conformity and Network Effects:** GWT applications are developed in Java which is a popular language, and can be developed with Java IDE and debuggers instead of Javascript debuggers. GWT can also benefit from the richness of existing Java APIs
- **Performances:** The JavaScript created by the GWT compiler performs well (Gmail is an example) and applications are cross-browser

Weaknesses

- **UI logic in Java:** Java is not the most suited language to lay out components in web pages (XML and scripting language based platforms are more intuitive for UI code when Java is more suited for procedural logic)

2.2.6 JavaFX

Description

JavaFX (or JavaFX Script) is a functional scripting language requiring to be compiled before being executed. This is the main difference with GWT and it places JavaFX as a direct competitor of Adobe Flex. JavaFX uses the JVM. It has been designed mainly to develop applications with strongly animated GUIs: Everything needed to make animations and manipulate graphical objects is present in JavaFX base APIs

Strengths

- **Skill-set Conformity and Network Effects:** JavaFX applications are developed in Java which is a popular language, and can be developed with Java IDE (there is an Eclipse Plug-in) and debuggers instead of Javascript debuggers. JavaFX can also benefit from the richness of existing Java APIs
- **Integration:** It is very easy to call Java code from a JavaFX script

Weaknesses

- **Lack of components:** There aren't many high-level graphical components available currently
- **Load:** As for Flash applications, JavaFX applications will take a few time to load on the first encounter
- **Penetration:** Java is less present than Flash Player (about 25% of machines don't have Java installed [[STAT](#)])

2.2.7 Echo3

Description

Echo is a component-oriented and event-oriented concurrent framework from GWT allowing to develop server-side applications (in Java with a Swing-like logic) as well as client-side applications (in Javascript). There is no compilation from Java to Javascript: during execution, the state of

components is serialized then sent to the client to display them with the Javascript library from Echo.

Strengths

- **Flexibility:** Allows the choice between server-side and client-side developing
- **Web languages Agnostic:** There is no need to know HTML nor even Javascript and CSS to design an Echo3 application

Weaknesses

- **No Network Effects:** There is a small community only; so there aren't that many high-level graphical components currently available

2.2.8 haXe

Description

haXe is a multiplatform open source language. It allows to choose the best target platform (Javascript, Flash, ...) for a given job. haXe provides the user with:

- A standardized language
- A standard library (the same on all platforms)
- Platform-specific libraries (APIs of each platform)

Strengths

Working across multiple platforms offers the following benefits:

- **Remoting between platforms:** Allows passing objects between different platforms
- **Single syntax:** Similar to Javascript and Actionscript
- **Conditional compilation:** Code logic can be easily transferred to the most suitable target in the platform

Weaknesses

- **No Network Effects:** There is a small community only; so there aren't a lot of specific libraries currently available
- **High Level:** haXe adds one more high level layer with one new language that won't take into account the differences between specific languages

2.2.9 Curl

Description

Curl is a reflective object-oriented language combining text markup (like HTML), scripting and heavy-duty computing (object-oriented like Java) in one single framework. Curl compiled applications are viewed using the Curl RTE, a runtime environment with a plug-in for web browsers.

Strengths

- **Performance:** It benefits from the advantages of a compiled, strongly-typed and object-oriented programming language.

Weaknesses

- **Penetration:** The Player has a very low penetration in comparison with other solutions
- **Skill-set required:** There is a short supply of skill-set because of the competition; so the learning curve is a risky investment
- **Toolset:** There aren't many high-level UI tools (such as good drag and drop) and graphical components

2.2.10 FlashiXML

Description

Following is a description of FlashiXML Website: [[FXML](#)]

FlashiXML is a rendering engine for UsiXML-compliant user interfaces (<http://www.usixml.org>) in a vectorial mode that is compatible with SVG (Scalable Vector Graphics is an XML-based specification to describe two-dimensional vector graphics). Any UsiXML-compliant user interface can be opened and rendered in this interpreter so as to create the truly working interface with presentation and dialog. In this environment, the UI can be resized at any time to address some constraints imposed by the computing platforms and to support some properties of Graceful Degradation of UIs, a sub-property of the Plasticity property. In this way, any UsiXML-compliant UI can be rendered on any computing platform equipped with a SVG or Flash plug-in or player.

For this purpose, an underlying mini-toolkit has been developed in ActionScript in the Macromedia Flash environment so as to render basic widgets which were not available natively in the Macromedia Flash environment.

Technological choices for RIA production

In this chapter, the three most important technological choices will be discussed:

- Why using ONME instead of another CASE tool
- Why choosing Ajax as a technique to create Rich Internet Applications
- Why using JQuery as a Javascript library

3.1 ONME

3.1.1 MDA and the OO-Method

The Model-driven architecture (MDA) supports MDE (see section 1.2.1 on page 6 for the definition of MDE) by involving models of the system at different levels of abstraction (Figure 3.1):

- The **Computation Independant Model** (CIM)
- The **Platform Independant Model** (PIM)
- The **Platform Specific Model** (PSM)
- The **Code Model** (CM)

The OO-method is MDA-compliant. It provides an explicit transformation mechanism between the levels of abstraction:

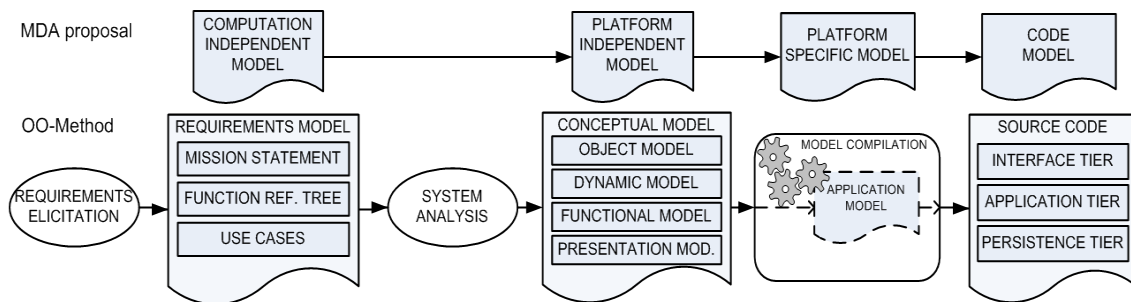


Figure 3.1: Correspondances between MDA and the OO-Method [P&a107]

The **Conceptual Model** is the equivalent of the MDA’s Platform-Independant Model. There are 4 views in this model:

- The **Object Model** defines the classes through which a program can examine and manipulate some specific parts of its world.
- The **Dynamic Model** defines the life cycles and the interactions of objects.
- The **Functional Model** describes the object’s state changes.
- Finally, the **Presentation Model** models the User Interface (UI).

3.1.2 Presentation Model

As explained in the section 2 page 9, the focus will be on User Interfaces (UI). So let's have a look at the structure of the Presentation Model of the OO-Method:

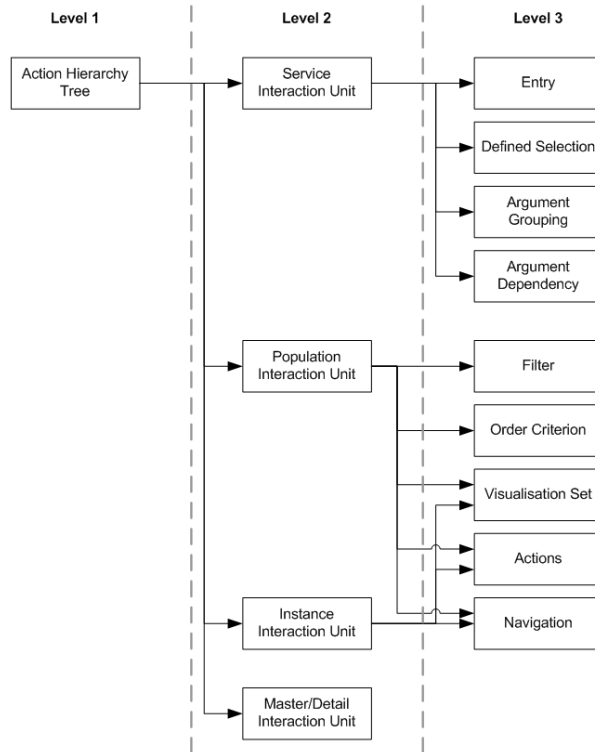


Figure 3.2: Decomposition of the Presentation Model [P&M07]

The Presentation model is made of three levels (enumerated from the most general to the most specific):

1. **Level 1: System Access Structure.** It describes the interactions with the system.
2. **Level 2: Interaction Units (IU).** The IU determine the scope of the interaction between the user and the system.
3. **Level 3: Basic Elements.** These patterns are the building blocs from which IU are created.

IU and basic elements are described in section 4.1 on page 26

3.1.3 Oliva Nova Model Execution

There are 58 commercial or open source tools that follow the MDA paradigm today [OMG09]. Most of them support UML modeling and other features of MDA such as transformation rules, model integration and code generation, but there are currently very few softwares that create source code for a target platform from the CIM.

Genova from Genera is a plugin for Rational Rose allowing dialog modeling and database mapping. Dialog modeling uses information from UML to generate a user interface proposal in the form of

a dialog model. This model can generate code for a given target platform such as Java, C++ or Visual Basic. Dialogs are combined to create applications. Those combinations are defined in the application model (following the OO-method paradigm). Changes of the Object Model regenerate dialog models and database mappings dynamically.

The official website of Genera (<http://www.genera.no/>) is unavailable and it became very difficult to find recent information on internet.

Jaxfront (<http://www.jaxfront.org/pages/home.html>) allows to generate graphical user interfaces on Java, HTML or PDF with an XML schema but does not follow the OO-Method.

Oliva Nova Model Execution (ONME) is a computer-aided software engineering (CASE) tool supporting the OO-Method. It offers important advantages as a candidate tool for the beautification process:

- It offers a full support of the OO-Method (the Presentation Model is well defined and it defines completely the system)
- It exports an XML file describing IU and basic elements with patterns (this will be explained in subsection 6.1.1 about the beautification process on page 35)

3.2 Ajax

We have seen the most popular asynchronous web programming environments in section 2.2 on page 12. Ajax has been chosen for the following reasons:

- There are lots of existing toolkits, components, and libraries (such as JQuery [JQ], JQuery UI [JQUI], Dojo (<http://www.dojotoolkit.org/>), Prototype (<http://prototype.conio.net/>), and the Yahoo User Interface Library [YUI]). They offer API at a higher level of abstraction and take care of lower-level implementation
- There are powerful platforms to develop Ajax applications. Aptana (<http://www.aptana.com/>) has been chosen because it is a complete web development environment that may be used as an Eclipse plugin
- Javascript language offers some similarities with Java so the skill-set needed was less important than with other technologies
- Debugging (which was one of the most popular original critics made on Ajax) is made easier thanks to firefox plugins Firebug and WebDeveloper. The last one allows seeing directly the generated source code of a web page, which is handy

3.3 JQuery

JQuery enables Unobtrusive Javascript: it allows to separate style from structure within an HTML document. JQuery makes common tasks trivials. The selector expression allows to identify target elements on a page to identify and grab needed elements easily.

Furthermore, JQuery UI offers rich UI widgets, such as the Accordion, the Slider and Tabs.

YUI library is used for widgets that weren't available in JQuery UI, such as the Carousel or the ImageCropper. YUI is developed by a known company, is well documented and some known websites such as LinkedIn (<http://www.linkedin.com/>) are powered by YUI.

Design patterns for RIA

The first part of this chapter describes ONME patterns. Then the results of the analysis of the applicability of fifteen RIA constructs will be presented here. Due to its length, the full analysis is available as an annex of this report.

4.1 ONME patterns

Basic Elements will be presented first, because they are building blocks for Interaction Units (IU). Then IU will be presented. The Action Hierarchy Tree is not present in the analysis because there is no information about it in the XML generated by OlivaNova, so it won't be considered for the beautification editor.

4.1.1 Basic Elements

- **Entry:** Corresponds to an input zone for a service. Data type, its format, valid values and a default value can be specified. Example: a textbox in a form
- **Defined Selection:** Corresponds to the selection of a value within a collection of valid values in a service.
Example: a combobox
- **Argument Grouping:** Allows to define groups for input arguments in a service. Those groups have aliases.
Example: a frame for textual information and another for various ID
- **Argument Dependency:** Allows defining dependency relationships between input arguments of a service with ECA (event-condition-action) rules.
Example: selecting an author in a defined selection will update books that can be chosen in another defined selection (showing only books written by that author)
- **Filter:** Corresponds to a query to get a subset of a population according to criteria.
Similar to WHERE in a SQL query
- **Order Criterion:** The ordering of a population according to one or more attributes.
Similar to ORDER BY in a SQL query
- **Display Set:** Attributes shown for a population or an instance.
Similar to SELECT in a SQL query
- **Available Actions:** Allows to invoke a service for an Instance or a Population IU. Privileges may determine who can access which services.
Example: When a book is consulted on Amazon, actions such as "Buy this book" are displayed to the user allowing to access the corresponding service
- **Navigation:** Allows to access a related object in an Instance or a Population IU.
Example: see the implementation in subsection 5.2 on page 31 displaying books related to a given book

4.1.2 Interaction Units

- **Service:** Allows to add, remove or modify properties of an object.
Similar to a form.
- **Instance:** Allows to manage an individual object.
Example: Displaying a book, its informations, showing services available for it and related objects
- **Population:** Allows the manipulation of a collection of objects.
Example: Displaying a list of books, mechanisms to select a subset and sort them, show related information and services available for them
- **Master-Detail:** Combination of the three other Interaction Units with a Master IU and Details IU.
Example: a population to choose a book and an Instance to display information on the chosen book.

4.2 Results of the analysis of RIA constructs for ONME

An analysis of 15 RIA constructs has been done by Francisco Javier Martinez Ruiz. This analysis is available in the first annex of that report. It is useful as a guide to understand constructs in the second analysis thanks to descriptions and facsimiles giving an idea of what the construct looks like.

Then an analysis of the applicability of these 15 constructs for ONME patterns has been done especially for this thesis. Professor Jean Vanderdonckt specified the legend (fig 4.1) of what needed to be specified for each (partially) applicable pattern and asked for a table resuming the applicability (4.2).

Legend	
Context of use	Specifies when, where, and how a RIA construct would be applicable in an ONME pattern, locally or globally
Constraints	Specifies the constraints to be satisfied in order to apply the RIA construct in the ONME pattern
Positive example	Provides an example where the RIA construct is properly, adequately applied in an ONME pattern (could be also a URL). Examples are imaginary unless specified and are applied to Amazon’s website (http://www.amazon.com) to make comparison easier
Negative example	Provides an example where the RIA construct is not properly, adequately applied in an ONME pattern (could be also a URL)
Usability value	Specifies the expected level of usability reached by applying the pattern (--, -, +/-, +, or ++)
Development cost	Quantifies the cost of developing the RIA construct for the concerned ONME pattern (very low, low, moderate, high, very high) NB. This can be multiplied if it is possible to implement the RIA construct in different environments, via libraries or APIs (e.g. Silverlight, jQuery)

Figure 4.1: Legend of tables of the analysis of applicability of RIA constructs for ONME

	Service	Instance	Population	Master- Detail	Entry	Defined Selection	Argument Grouping	Argument Dependency	Filter	Order Criterion	Display Set	Available Actions	Navigation
Starfield	-	-	++	++	-	-	-	-	++	-	-	-	-
Accordion	-	++	-+	++	-	-	++	-	-	-	++	-+	++
Carousel	-	-	++	++	-	-	-+	-	-+	++	-+	++	++
ImageCropper	-	-+	-	-	-	-	-	-	-+	-	-+	-	-
MovingCovers	-	-	-	-	-	-	-	-	-	-	++	-	++
Mosaic	-	-	-	-	-	-	-	-	-	-	++	-	++
ImagePuzzle	-	-	-	-	-	-	-	-	-	-	++	-	-
AlphaSlider	-	++	++	++	-	++	++	-	++	-+	-+	++	++
Dock bar	-	-	++	++	-	-	-	-	-	-	-	++	++
Clustermap	-	-	-	++	-	-	-	-	-	-	-	-+	++
3D Cloud	-	-	++	++	-	-+	-	-	-+	++	++	++	++
Zoom Menu	++	-	-	++	-	-	++	-	-	-	++	-+	++
Dashboard	++	-	-	-	-	-	++	-	-	-	-	++	++
Tag clouds	-	-	-+	++	-	++	-	-	-	-	-	-	-
TimeLine	-	-	++	++	-	-	-	-	-	-	++	-+	-

Figure 4.2: Table of applicability of RIA constructs for ONME (++: Applicable, +/-: Partially applicable, -: Not Applicable)

For each construct, its applicability for a pattern was assessed thereby:

Look for existing scripts to have an overview of features provided by that construct and potential uses.

Determining whether this construct could be applied for this pattern. Think about an example where it could fit. If there is an example,

- Think about a negative example to have a first idea of limits of applicability
- Specify as completely as possible the context of use of such a construct for this pattern
- Assess the applicability and determine whether it is fully applicable or partially applicable. It is partially applicable if the pattern’s features are restricted using that construct (for instance if there is no Order Criterion for a Population, or if a Filtering process has to be done manually instead of using a query)
- Identify intrinsic constraints of using that construct and, if possible, constraints of using it with ONME (for instance interaction required from the user)
- Finally, evaluate the usability (for the user) and the complexity of implementation to have an idea of which to implement first

To make examples easier to understand, they were used if possible in a same context (an on-line retailer such as Amazon (www.amazon.com)).

The help of an expert of ONME (Nathalie Aquino) allowed defining what was actually possible to implement with an in-depth knowledge of transformations.

The work has been consolidated by Francisco Javier Martinez Ruiz, who invalidated some cases based on the feedback of Nathalie Aquino with an explanation of what they couldn't be applied for beautification. Those cases are labeled in red in the table. He also included the Domain Model and the brief explanation of OlivaNova patterns, the general case for the Context of Use, added the analysis of the Timeline and did some local additions to the tables.

The last step to make all examples understandable was to make drawings of most complicated constructs. The following patterns were considered as hard to understand:

- **Starfield Display:** Multiple dimensions are expressed with tracks, the ruler and coding schemes
- **Clustermap:** Markers have various uses (for instance as regions)
- **Zoom Menu:** The number of levels of interaction and the navigation between menus is hard to understand
- **Dashboard:** Content and display of windows may be hard to understand (for instance if the minimized window represented by a label)

In addition, this document was read by someone without an IT background to identify individual examples hard to understand and add some additional drawings:

- **Accordion for Argument Grouping:** To understand how it fits in a service
- **Carousel for Define Selection:** For the same reason
- **Carousel for Display Set:** To understand how it interacts with the display

Implementation of significant RIA constructs

This chapter presents manual implementations of some RIA constructs from the analysis.

Here are the motivations for manual implementations:

- **Learning Web Development languages and techniques:** Developing web applications requires a specific experience and shows some differences with desktop applications. Examples below have been chosen specifically to learn some languages and techniques.
- **Have an overview of some existing constructs:** Some constructs are made for a specific use that could not fit with the expected use for some patterns of OlivaNova. Implementing a manual example allows to see for which pattern it is effectively suited. This may often not be understood with the documentation provided with the library: some properties may not be alternable. Using the construct in a specific context may also have bugs.

5.1 Master-detail with a slider and an accordion

Here are the motivations for choosing this example:

- **Discover HTML:** Using multiple constructs that need to be displayed efficiently with a table and headings tags
- **Discover Javascript:** Using external scripts, Javascript Arrays and loops to iterate through them, handling strings (replace characters)
- **Discover JQuery:** Using selectors to get and update values
- **Test JQuery UI constructs:** Both constructs present in the analysis (accordion and slider) are used in this example
- **Apply an Interaction Unit:** The next examples implement basic elements but this one implements the master-detail to use both constructs while not being constrained by the choice of a basic element the construct is suited for: constructs are used for their original purpose here:
 1. Selecting an object with the slider, one of the features of the Population Interaction Unit
 2. Displaying attributes of the selected object and related objects with the accordion, two features of the Instance Interaction Unit

Please choose your book in the slider below:



Chosen book: **Isaac Asimov - Foundation**

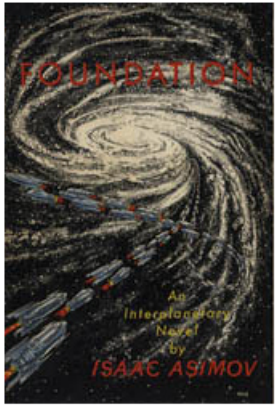
	Publisher
	Year
	Price
	Related Books
	Isaac Asimov - I, Robot Arthur C Clarke - Rendezvous With Rama

Figure 5.1: Master-detail with a slider and an accordion

Book attributes were stored in a CSV file. An online converter (<http://www.creativyst.com/cgi-bin/Prod/17/eg/csv2js.pl>) was used to convert values to Javascript Arrays. Related books are determined manually in Arrays. Images are retrieved from the directory Images with their names (as some characters such as question marks or columns can't be used in file names, they have not to be taken into account).

Using JQuery , the slider dynamically updates information in the accordion and the image of the book displayed with values corresponding to the element selected.

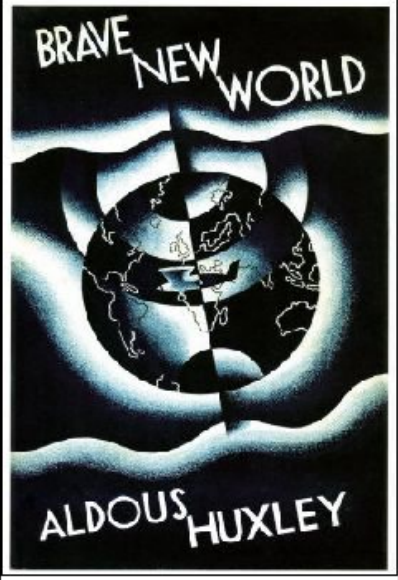
This example allowed to see that the accordion widget animation may cause troubles because the size of the accordion changes abnormally during the animation. The slider is basic and the selection of values by clicking is not really precise, but works when the cursor is moved so it can be applied for the beautification editor.

5.2 Carousel for navigation basic element

Here are the motivations for the choice of this example:

- **Getting more familiar with Javascript:** Discovering the global scope of variables in functions and recursivity
- **Discover the link between Javascript and the DOM:** Dynamically creating HTML tags in the document with Javascript
- **Discover CSS:** Setting properties of the Carousel with Cascading Style Sheets

- **Test a construct from YUI:** As seen in the analysis, the Carousel is a construct applicable to many patterns so it was one of the most interesting constructs to test

	<p>Displayed book: Aldous Huxley - Brave New World</p>	<p>Publisher: YHN</p> <p>Year: 1932</p> <p>Price: 7</p>
--	---	--

Here are some related books:



Figure 5.2: Carousel for navigation pattern

The same Arrays with book information as those of the first example are used in this one. An image of the book and information about it are displayed in a table. A Carousel below displays related books. The interesting part of this example is that when a user clicks on a related book, the information in the table is updated, but related objects of the new selected book replace the old ones in the Carousel. The function *newCarousel* is called when a related book is selected. For the first related books, a new context of scope has to be created because the scope of variables is global to the function:

```
image.onclick = function(j){
    return function(){
        newCarousel(j);
    }
}(Related[book][i]);
```

To update the Carousel without refreshing the page, Carousel elements need to be changed dynamically:

```
carousel.clearItems();
var i = 0;
while (Related[k][i]) {
    carousel.addItem("<img src=\"" + Images[Related[k][i]]
+ "\" height=\""150\" width=\""100\" onclick=\""newCarousel("
+ Related[k][i] + ");\" >");
    i++;
}
```

There was a bug in one of the CSS files related to the Carousel widget provided in the package of the second beta of YUI but the file hosted on yui.yahooapis.com was working. This detail aside, the widget is working as intended so it is the primary widget choice for the prototype.

5.3 Imagecropper for Available Actions basic element

Here are the motivations for the choice of this example:

- **Getting more familiar with Javascript:** Retrieving values from the HTML, using Image objects and handling events
- **Getting more familiar with HTML:** Using forms and buttons, using styles
- **Test a construct from YUI:** This is another YUI construct present in the analysis. The third and last one was a slider. This construct was already implemented with JQuery UI.

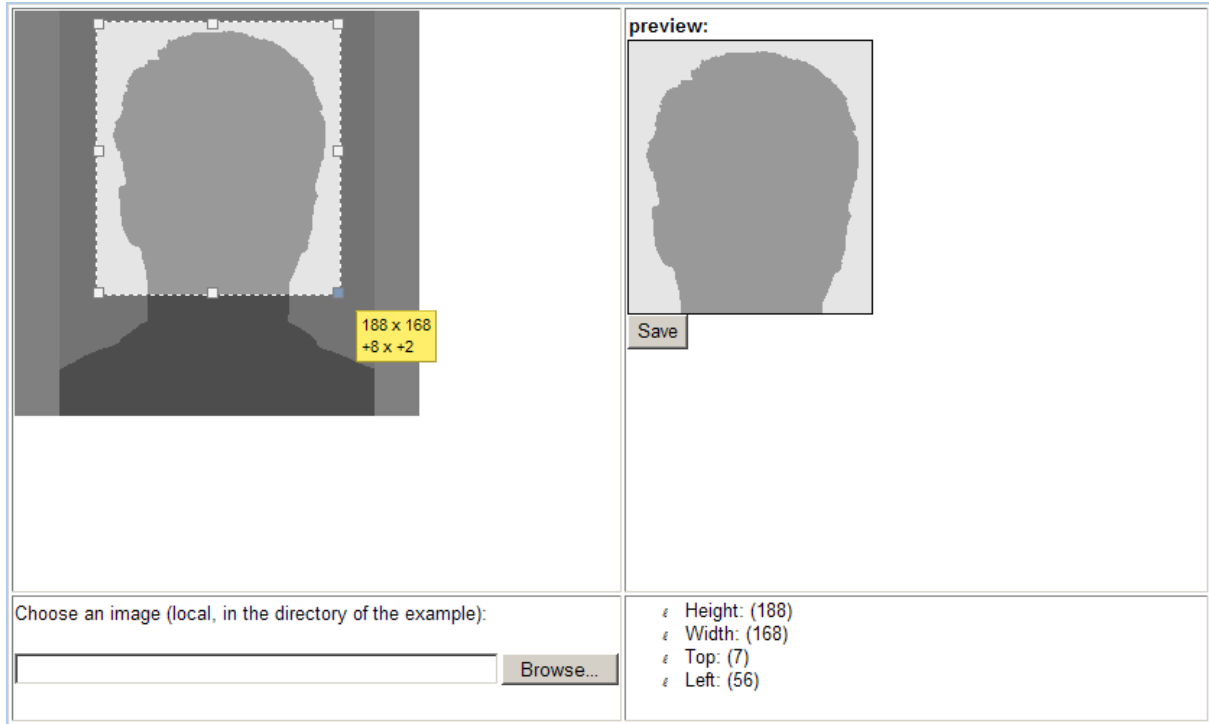


Figure 5.3: Imagecropper for Available Actions pattern

An Imagecropper allows selecting an area on an image. A preview of the selected area is displayed too. The cropped image attributes (height, width and position of the upper left corner) are displayed and updated on the event *moveEvent*. A form allows to upload a local image. The save button only allows to lock the values of the cropped image because there is no client/server interaction.

This construct has a limited use (for instance for this pattern it allows to execute only one service: resizing an image) and it can't really be used to improve a pattern in a classical application generated by OlivaNova such as the Expense Reports application.

Development of a software to support RIA production

This chapter describes the design and the implementation of the beautification editor prototype. Starting from what was explained before on OlivaNova and the OO Method in general, a first step will explain where the tool is located in the transformation process from models to code.

Then the specifications for the tool of this MSc thesis will be explored: the original expectations for the required tool will be discussed.

The third section describes features that have actually been implemented in the prototype.

The fourth section gives a thorough overview of the packages as well as classes and sequence diagrams.

Finally, an estimate of the effort of the implementation of the software will be given.

6.1 Software approach

The Model-Driven Approach has been explained in subsection 3.1.1 on page 22 and the motivations for a beautification process in Model-Driven Engineering of User Interfaces have been described in subsection 1.2.2 on page 6.

Knowing that information, the beautification process for User Interfaces in OO-method can be explained more thoroughly.

6.1.1 Beautification process

The beautification process is made of three steps [P&a107]:

1. **Derivation of a Concrete User Interface Model from the Presentation Model (fig 6.1):** A Concrete User Interface is an abstraction of the Final User Interface without taking into account specific widgets of target platforms. The User Interface is characterized in terms of Concrete Interaction Objects (such as a combo box or tabs) with attributes.
2. **Application of the Beautification Operations (fig 6.2):** The goal of the beautification is to modify existing Concrete Interaction Objects, not delete or create new ones. A User Interface Editor detects which beautification operations can be applied to Concrete Interaction Objects and allows to select the desired operation to apply. It gives then a preview of the modification. Parameters are used to modify values of components of the User Interface.
3. **Generation of the Final User Interface (fig 6.3):** When all beautification operations have been made, the Concrete User Interface Model is updated and transformed into code with the other models. To achieve that, the Model Compiler needs to be able to understand

the new Concrete User Interface Model with the modifications brought by the beautification editor.

The first step is done by the Oliva Nova Model Execution System, which exports an XML file describing Concrete Interaction Objects in form of Patterns (Interaction Units and Basic Elements of the Presentation Model, which is a User Interface Abstract Specification Model).

The purpose of the software is therefore to achieve the second step. The third step can only be performed after completion of the second and requires modifications of the Model Compiler.

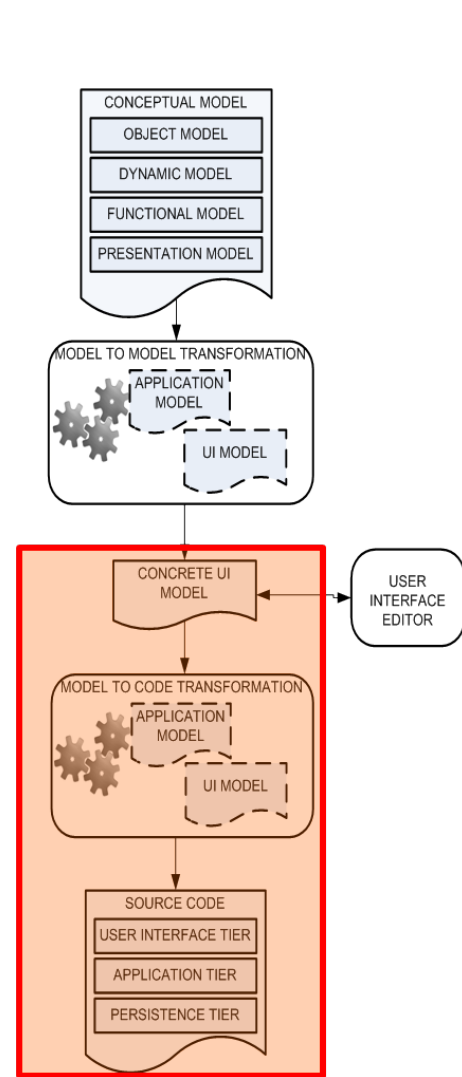


Figure 6.3: Generation of the Final User Interface [P&a107]

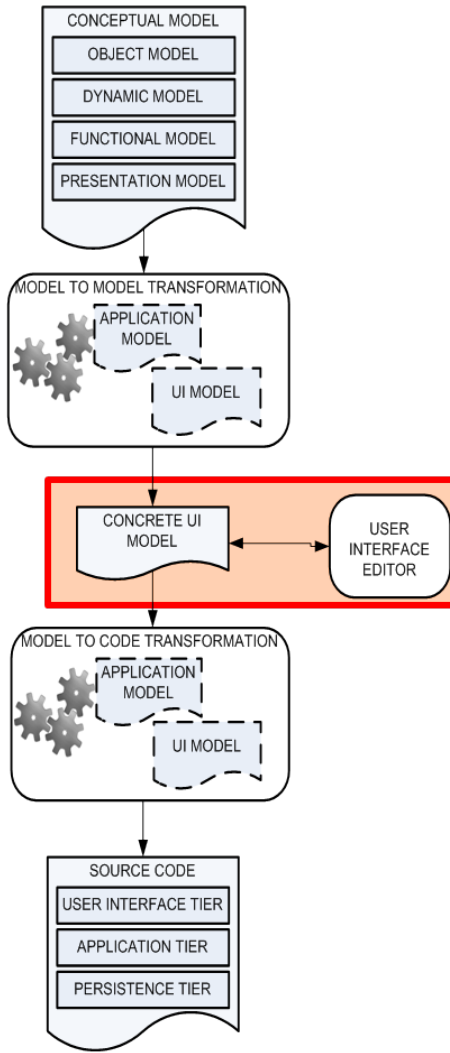


Figure 6.2: Application of the Beautification Operations [P&a107]

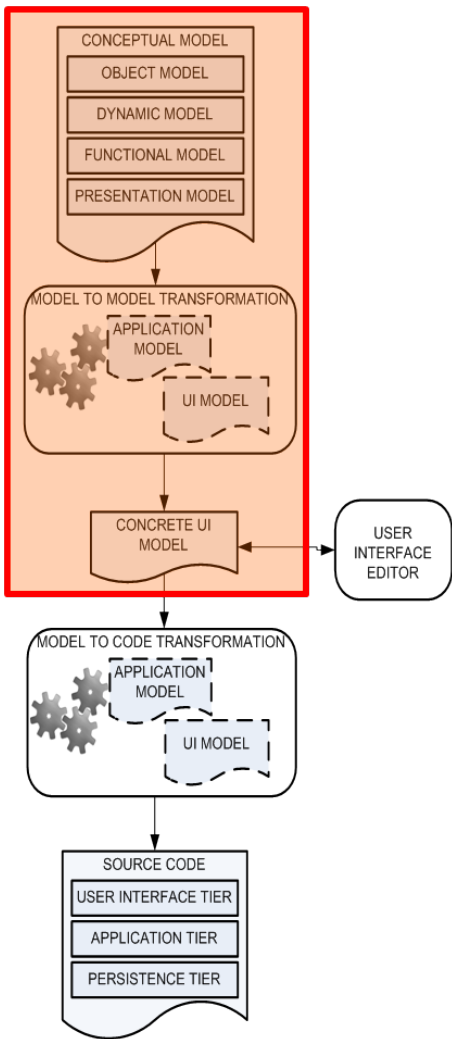


Figure 6.1: Derivation of a Concrete User Interface Model from the Presentation Model [P&a107]

6.1.2 Beautification editor for ONME

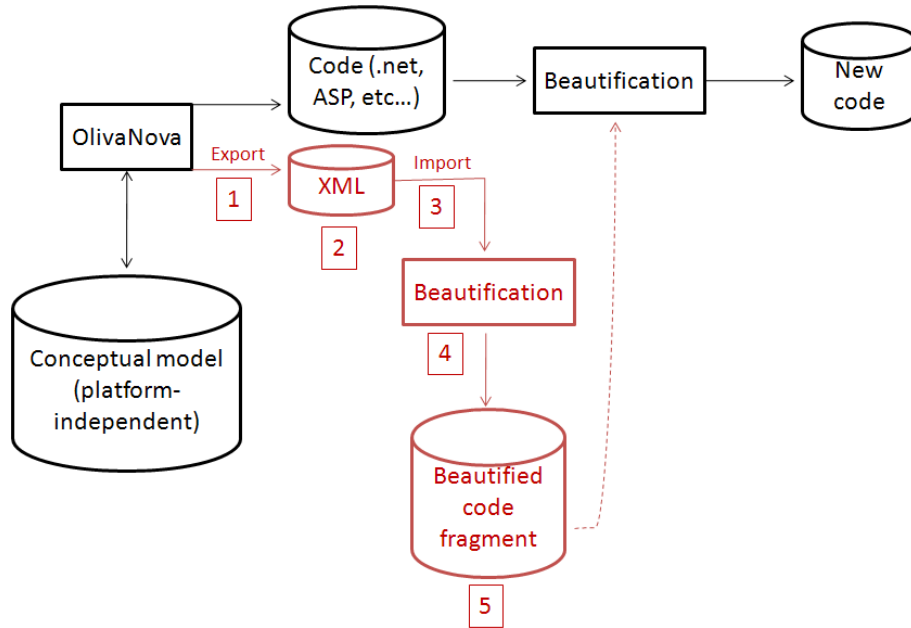


Figure 6.4: Beautification process covered by the editor

Figure 6.4 gives an overview of the process covered in this MSc thesis (pictured in red). We will explain each part in the following sections

1. Exporting an XML file from an OlivaNova project
2. Browsing through the XML file to identify patterns
3. Importing Patterns of interest into the editor
4. Allowing the user to select the desired modifications to apply and give a preview of the modification
5. Generate fragments of beautified code

The dotted red arrow represent the export of a new Concrete User Interface Model to the Model Compiler in order to produce code for the whole project with beautification modifications. While it corresponds to the third step of the beautification process, Oliva Nova Model Compiler is currently unable to take these modifications into account so this part wasn't covered in this thesis.

6.2 Specification of the tool

It is now possible to identify what were the expectations for the tool.

The goal diagram shows that the tool process should be sequential, allowing two kinds of user interaction for pre-made beautification operations. It also gave an intuition on an interaction scenario:

- **XML importation:** The user chooses an XML file exported by an OlivaNova project

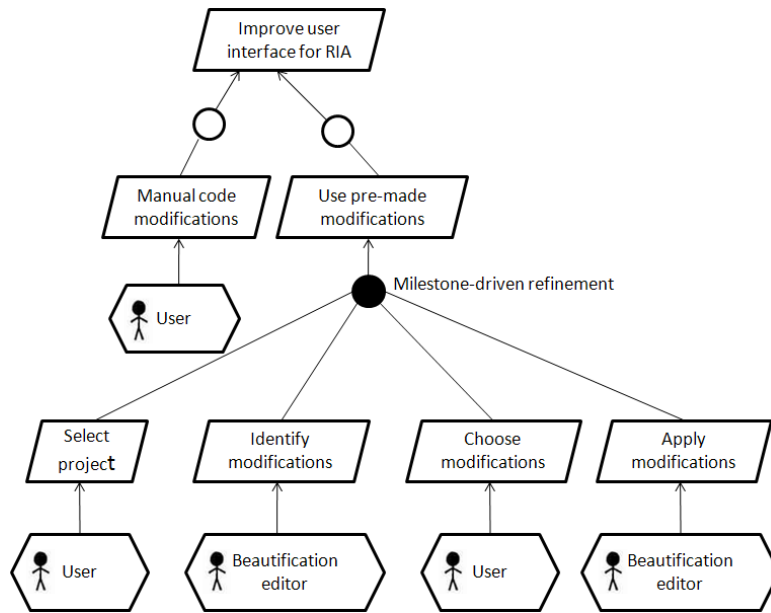


Figure 6.5: Goal diagram of the beautification editor

- **XML parsing:** The software identifies all patterns eligible for beautification regarding an RIA application and stores them

When all eligible patterns are identified, the tool repeats the following steps for each occurrence of them:

- **Choose beautification operations:** The user chooses the widget he wants to apply to the occurrence of the pattern or skips this occurrence
- **Choose parameters (optional):** The user may want to choose parameters for the selected widget such as the size or the orientation
- **Generate a preview:** The user needs to have a comparison between the default display and the beautified widget he has chosen
- **Save beautified code:** When the user confirms his choice, the beautified code is saved and the following occurrence is processed

6.3 Implemented features

This section describes features allowed by the tool. It can also be used as a guide explaining how to use it.

The tool is developed in Java. It uses the system look and feel and automatically fits to the window size when it is launched. A File Chooser is initially opened to select an exported project. A Text Area in the bottom gives information on the process advancement to the user. Initially it specifies the kind of XML file the user is expected to load. This is particularly necessary because there are two different kind of XML files exported by OlivaNova, which already led to confusion:

there is a Multilanguage export (.mlg.xml files) that seems to be suited for external tools such as the beautification editor but is actually a less complete version of the Object Oriented Model export (.oom.xml files), lacking key parameters for widgets such as the service type in the Available Actions pattern (explanation later in the text).

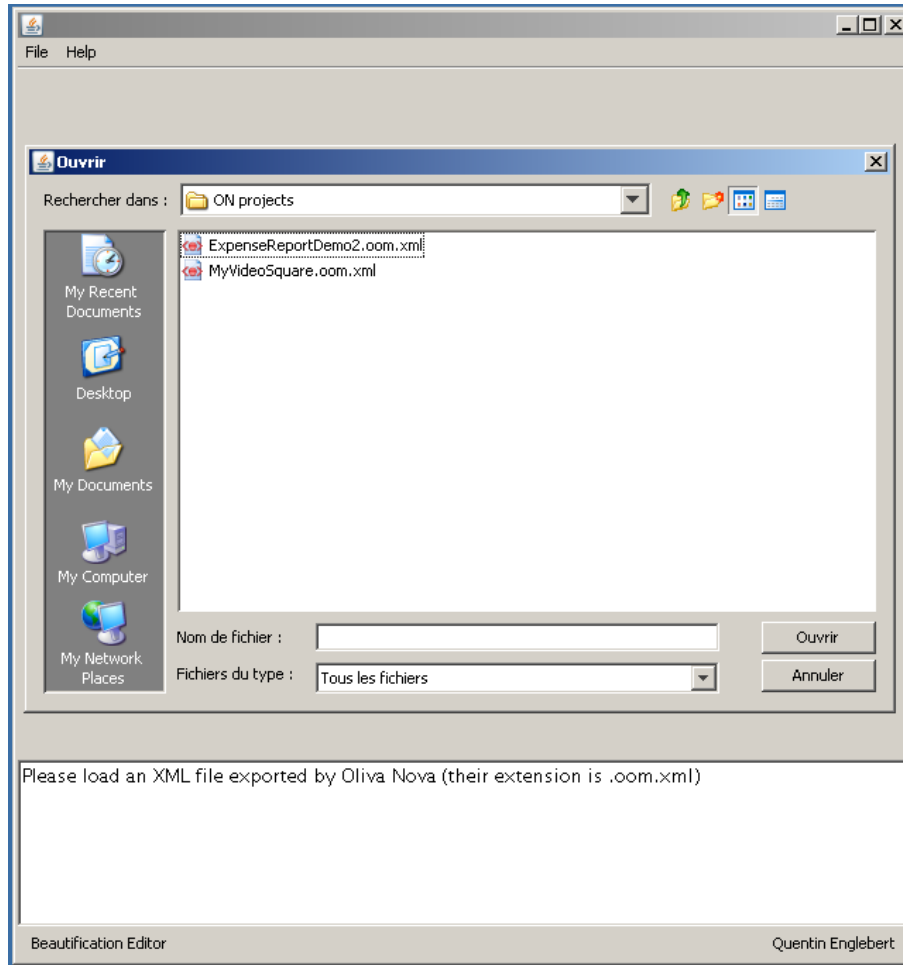


Figure 6.6: Launching the editor

After selecting an XML file, the Menu Bar becomes available, allowing to choose another XML file, close the tool or have more information about the author. The document is quickly parsed, the number of occurrences detected for each pattern is displayed. The panels for user interaction are displayed:

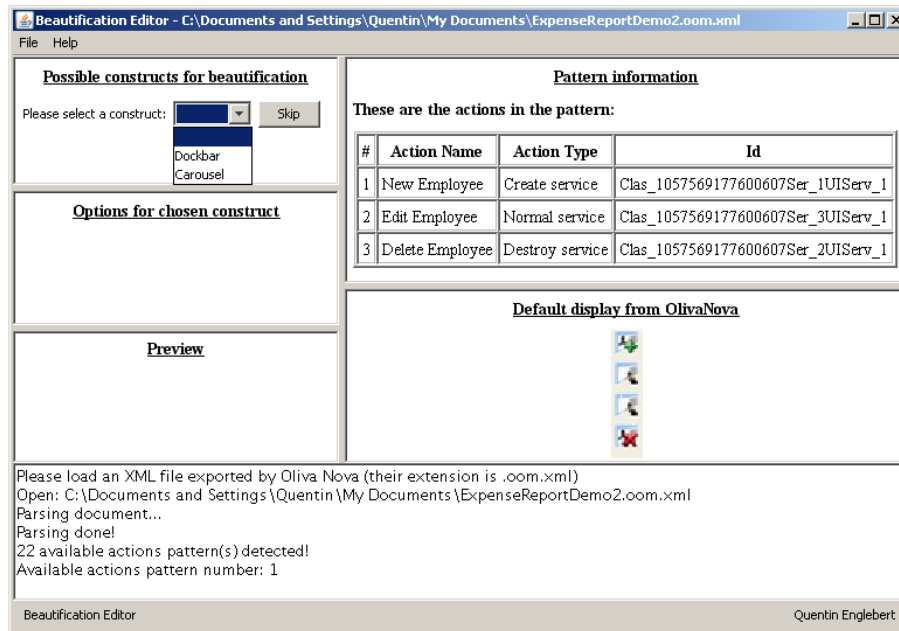


Figure 6.7: Parsing done

The panels are displayed using the MultiSplitPane Swing Container is used. It allows a resizable tiled layout of components without nesting. The layout is defined with a simple tree-structured model that can be stored and retrieved to make the user's layout configuration persistent. [MSP]

Here is the Layout Definition for the panels in the tool:

```
// Definition of the MultiSplitLayout to display the four panels
// consisting of two columns sharing the screen width and resizable
String layoutDef = "(ROW (COLUMN cc opt p) (COLUMN ip on))";
```

As seen on figure 6.7, three of the five Panels contain directly information:

- The Information Panel (ip), labeled "Pattern information" displays information on items of the pattern to help the user understand which occurrence of the pattern is currently eligible for beautification
- The OlivaNova Panel (on), labeled "Default display from OlivaNova" displays a screenshot of the default look of the pattern. It is not possible to generate the actual transformation to give a preview because OlivaNova code transformation is done by a server and the generated code is sent by mail. The benefit of implementing the same transformation with this tool is low for a big implementation effort: giving a screenshot of the result of a transformation with information on the current pattern is enough to understand what the default OlivaNova generated pattern looks like in the context of a prototype.

- The ConstructChoice Panel (cc), labeled "Possible constructs for beautification" allows the user to choose the widget he wants to apply to the current pattern with a contextual menu. He may skip to display the next pattern occurrence too.

The panels on the right side of the screen are designed to provide information, while the panels on the left side require interaction. When the user selects a widget, the content of the two last panels is updated:

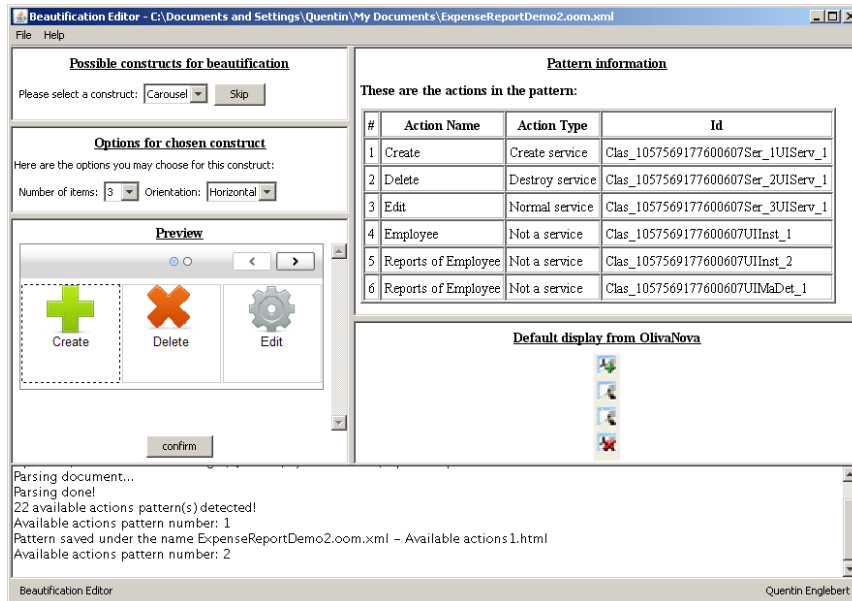


Figure 6.8: Chosen construct

- The Options Panel (opt), labeled "Options for chosen construct" allows to change parameters related to the selected widget. In this example, the Carousel widget is chosen and the user may change the number of items displayed at a time and select the orientation of the Carousel.
- The Preview Panel (p), labeled "Preview" displays a preview of the generated RIA widget with chosen (or default) parameters in a JWebBrowser [JWB]. This browser based on Mozilla avoids the need to display the construct in an external browser, easing the comparison between beautification operations and default OlivaNova transformation. Using an external library was necessary because Java doesn't interpret Javascript by default. The widget is generated in a HTML file with the name of the project, the name of the pattern and the occurrence number (for instance "ExpenseReportDemo2.oom.xml - Available actions2.html"). When the user confirms his choice, the HTML file is kept and the next occurrence of a pattern is displayed. If he chooses to skip the occurrence, the HTML file is deleted.

While XSLT transformations were initially planned to generate beautified widgets, the generation isn't performed from an XML file anymore, but from some data that have already been parsed from the XML and parameters given by the user. Generating the HTML file from a default Java String with inclusion of some parameters, values or specific lines of code at specific places is now both handier for the developer and faster for the tool.

6.4 Implementation Architecture

Due to time constraints, the tool was first done with the basic idea of a prototype: implement one or some example operations to validate the beautification process. When the first construct was implemented for the first pattern and the base goal was actually met, it was interesting to facilitate the addition of new patterns and/or constructs. So a code refactoring was done with this goal in mind: make the maximum amount of processing steps generic and only have to add specific code where it is necessary.

6.4.1 Packages

To have a better overview of the architecture, we will first explore the highest level diagram: the packages diagram.

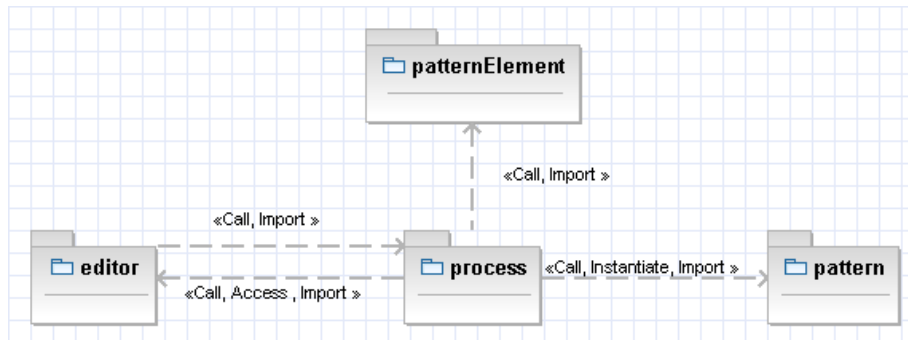


Figure 6.9: Packages diagram

There are four packages:

1. **editor**: contains the main class and classes related to the GUI
2. **process**: contains the XML parser class and classes to process beautification operations
3. **pattern**: contains classes corresponding to detected occurrences of patterns
4. **pAtternElement**: contains classes corresponding to elements of the occurrences of patterns.
For instance, an occurrence of the Available Actions pattern is made of Actions elements.

The content of these packages will be detailed in following subsections.

6.4.2 Classes

This subsection contains class diagrams for each package. To have clear diagrams, only the relations between classes of a same package have been drawn. The relations between packages will be explained in the text.

Editor

The main method is in the class BEditor. It calls the invokeLater method from Swing, the Open method for the JWebBrowser and creates an instance of BEditor. The constructor creates an

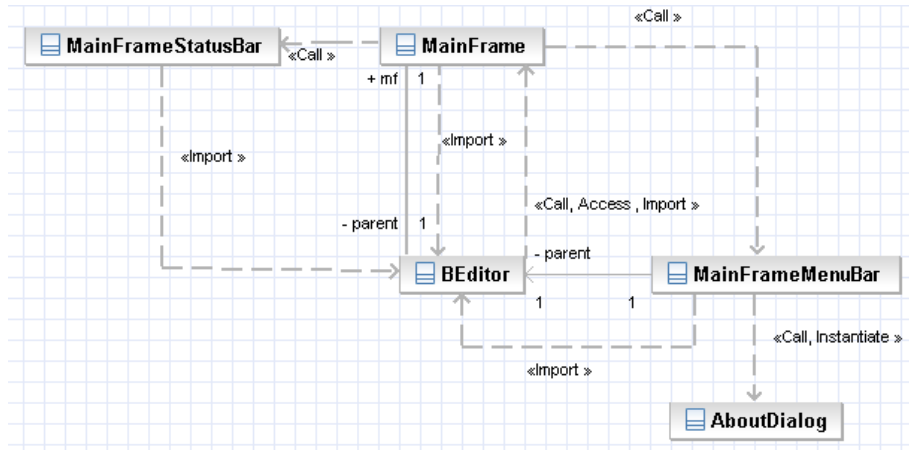


Figure 6.10: Class diagram of editor

instance of the MainFrame then calls openXMLFile. As seen on figure 6.11, this method allows to choose an XML file, creates an XML parser, launches the parsing of the document, creates a process object when the parsing is done and launches the process of the first pattern.

MainFrame creates the Main Frame, creates the Menu Bar (creates an instance of MainFrameMenuBar), the Debug Area (the text area in the bottom of the screen) the Status Bar (creates an instance of MainFrameStatusBar, this bar is the bar under the Debug Area) and the Transform Panel (its content will be updated by Process).

MainFrameMenuBar creates the context menu and creates an instance of the AboutDialog (displaying information about the authors in HTML in a JEditorPane with mailto links).

Figure 6.12 lists attributes and methods of each class of the package editor:

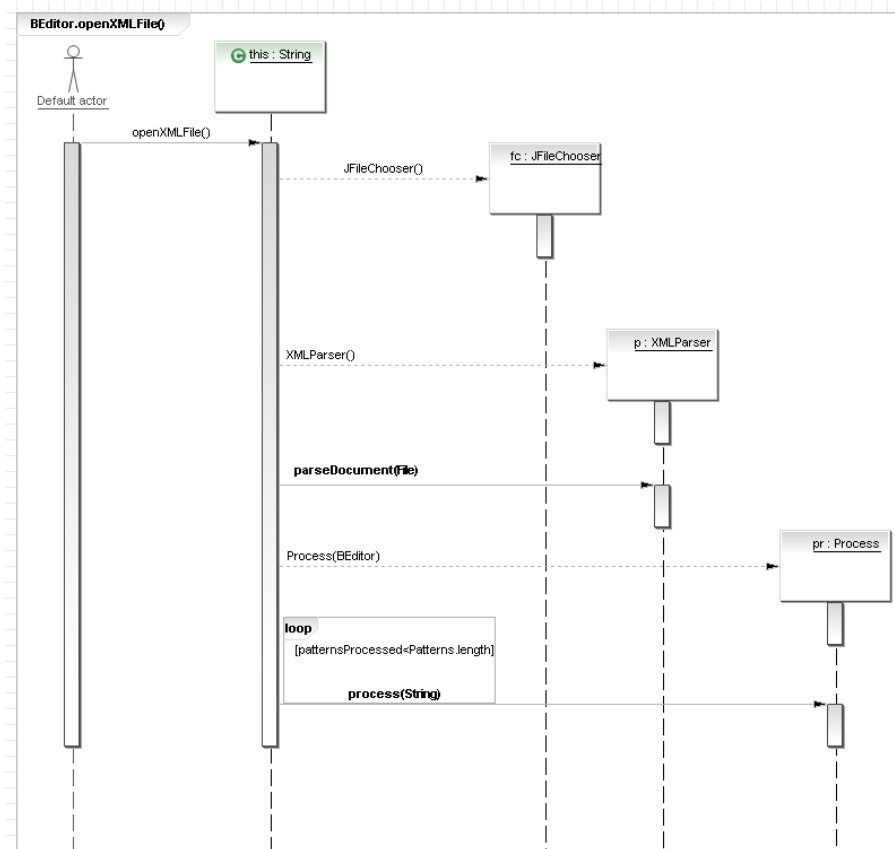


Figure 6.11: Sequence diagram for openXMLFile

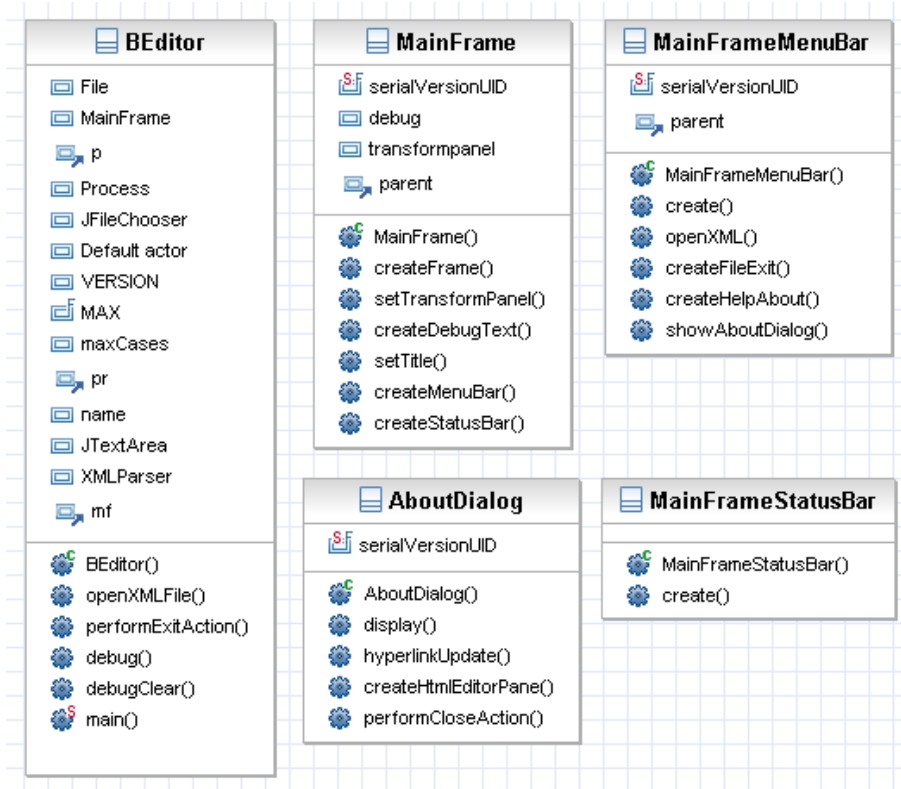


Figure 6.12: Classes of editor with methods and attributes

process

As said in the beginning of the section, everything is made in a generic way and the process is different only when it depends on the pattern. The class Process is the biggest class. It creates the five panels of the Transform Panel, sets and update their content. This class may seem monolithic with a lot of global variables, but it is intended to provide genericity: nothing in it has to be changed to handle new Patterns or new Constructs.

When it needs to display elements depending on the pattern, it will call generic methods in ProcessGen that will check the type of the current method and call the suited method. Figure 6.14 is an example: to create the ComboBox displaying widgets available for current pattern, the Construct Choice Panel calls *makeComboBox* on ProcessGen. It will check the type of the pattern and call the right process (here *makeSpecComboBox* on ProcessActions).

Other similar methods are *getIcon* to display the default Oliva Nova display image for the Oliva Nova Panel, *makeOptions* to display the options available for the chosen pattern, *setTextPatPanel* to display the table of information on the current pattern and obviously *createConstruct* to create the selected widget with chosen options.

So the goal of ProcessGen is to detect the type of the pattern and call the right method on Process[name of pattern]. Each of those specific classes have to implement the five similar methods listed above, defined in the interface ProcessSpec. These methods return a GUI Component or a

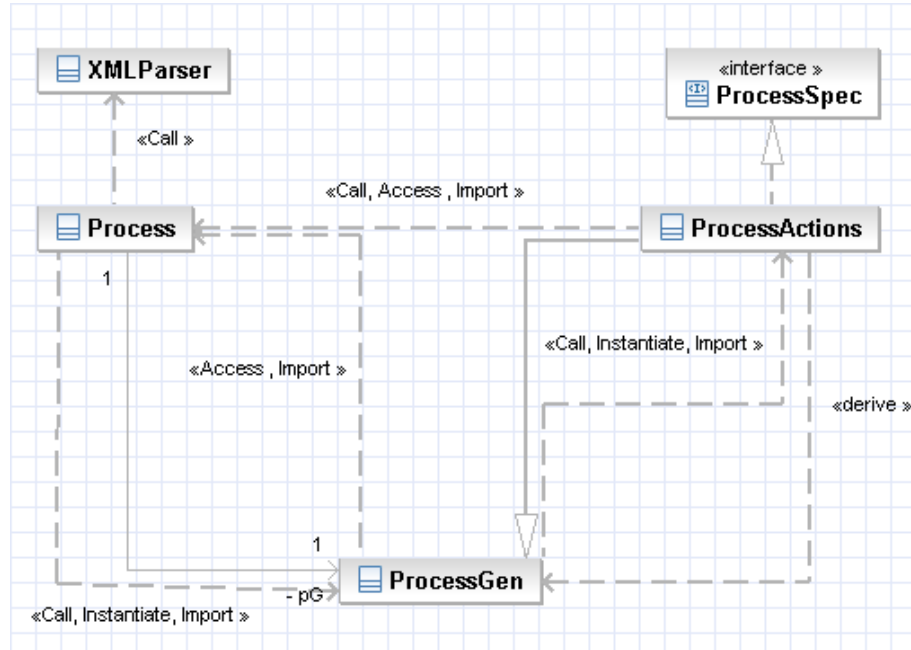


Figure 6.13: Class diagram of process

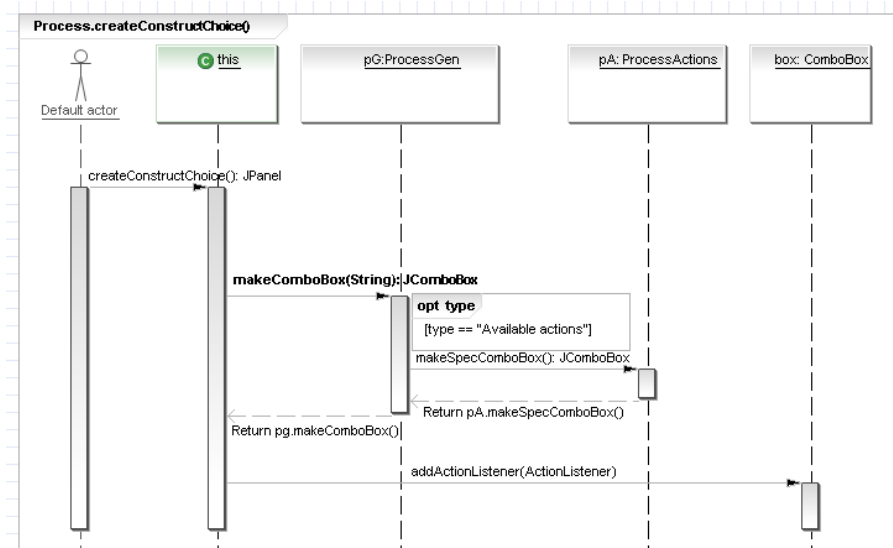


Figure 6.14: Sequence diagram for Choice Panel

String and may call methods of Process to update their panel.

The class XMLParser is the only class that has not been made generic, because the parsing should be done only one for efficiency. It should only be modified for the inclusion of a pattern, which is less frequent than including a new construct (because there is a limited amount of OlivaNova patterns). The XMLParser creates an ArrayList of patterns occurrences for each pattern. Thanks to SAX, it parses the document only once and tags opening, closing or text between tags may trigger some action.

For instance, for the Available Actions pattern, it detects an occurrence of the pattern, adds a new instance of ActionsPattern the ArrayList of Available Actions, and adds all actions of that occurrence to the ActionPattern object with their ID and their alias.

In parallel, the parser detects each service and adds their type to an Hashtable with their ID as a key. When the parsing is done, all keys corresponding to actions parsed for the Available Actions pattern are checked to couple the type of service with the action. This allows to define an Action as a triplet as shown in the table: ID, alias, type. There is also a generic getter methods for the Process class to get the ArrayList of occurrences for a type of pattern.

Figure 6.15 lists attributes and methods of each class of the package process:

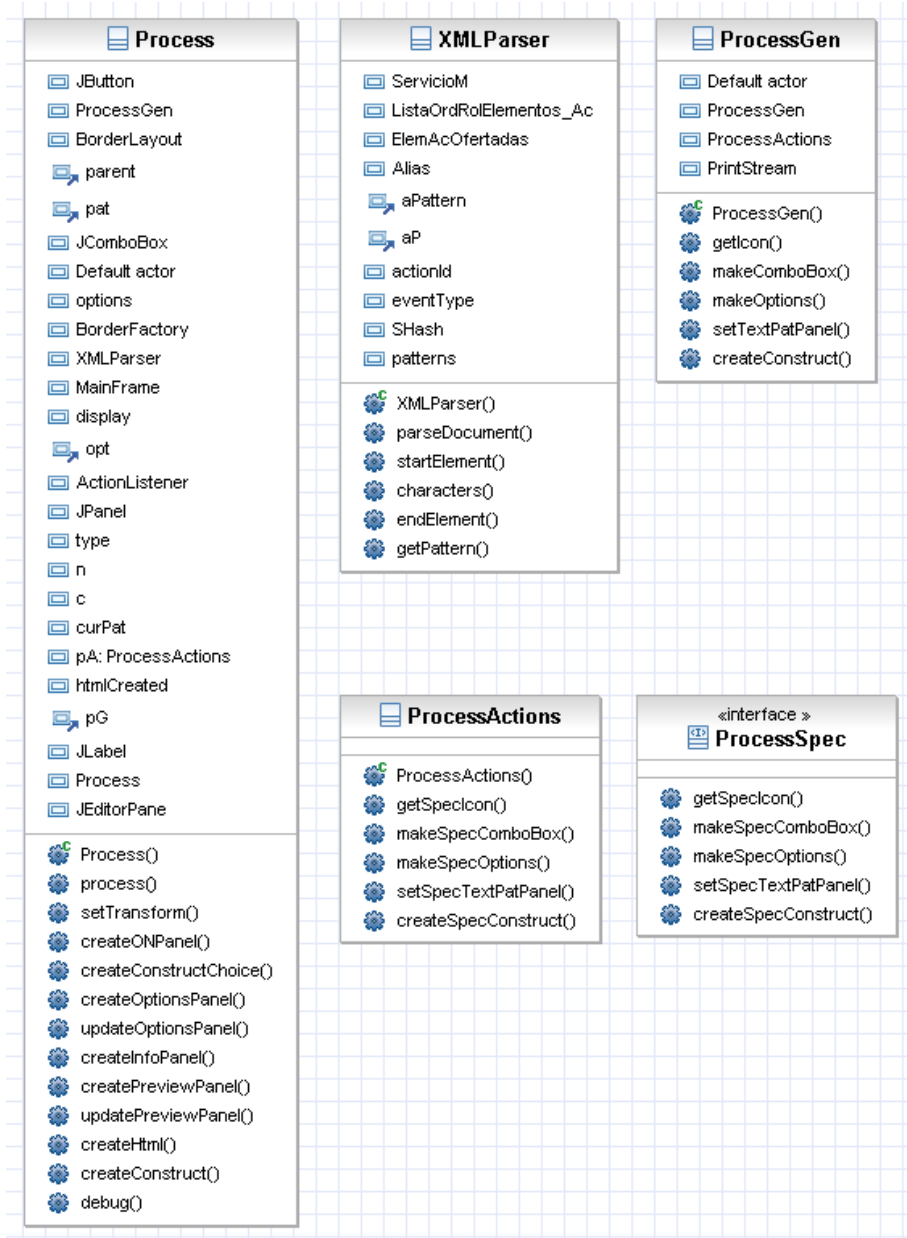


Figure 6.15: Classes of process with methods and attributes

pattern

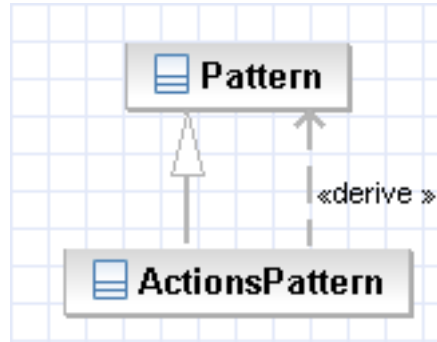


Figure 6.16: Class diagram of pattern

Each type of pattern needs to have a class [Pattern name]Pattern. Those classes extend the class Pattern (only sharing a variable type). Their methods are called by the XMLParser to get the list of elements of an occurrence of a pattern or to add an element to the ArrayList of elements of an occurrence. There is no interface for specific pattern classes because their methods may be different: the ArrayList of each pattern contains different PatternElements objects (here Action objects) so the *getList* method is different and the *add* method depends on attributes of elements. For instance, we've seen that an Action object is defined by a triplet of values and only two (the ID and alias) are added upon detection of the pattern.

Figure 6.17 lists attributes and methods of each class of the package editor:

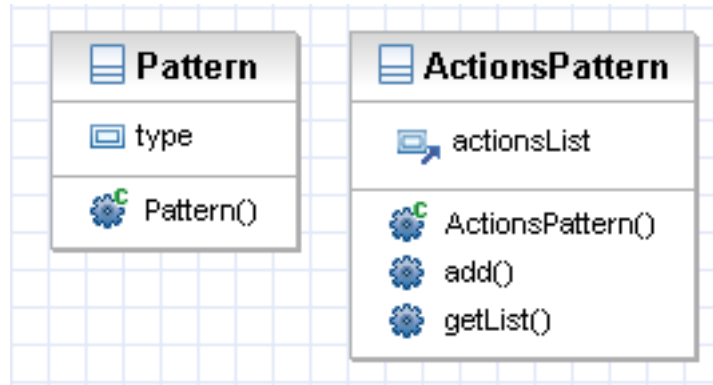


Figure 6.17: Classes of pattern with methods and attributes

patternElement

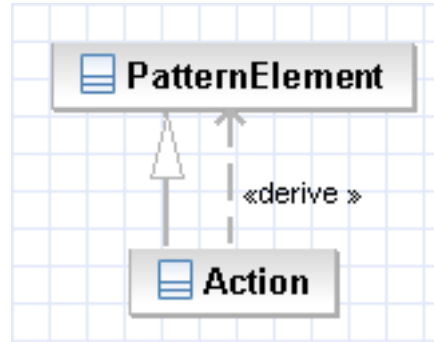


Figure 6.18: Class diagram of patternElement

Each type of PatternElement needs to have a specific class, such as the Action class. Those classes extend the class PatternElement (only sharing a variable type). Their methods are getters and setters for the attributes used by the XMLParser and their specific Process class.

Figure 6.19 lists attributes and methods of each class of the package editor:

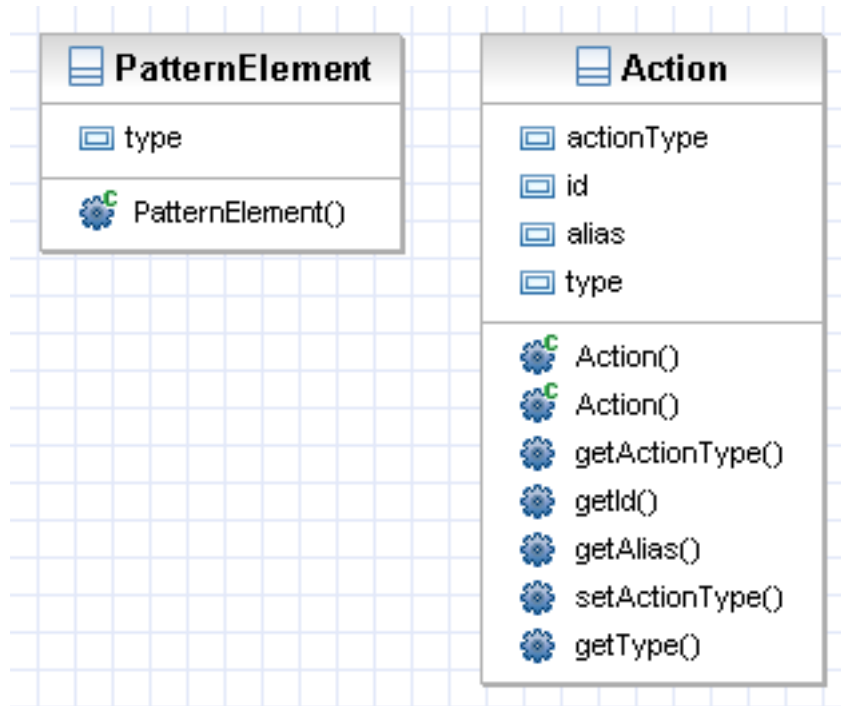


Figure 6.19: Classes of patternElement with methods and attributes

6.4.3 Adding a widget

Adding a widget such as the dockbar is now fairly simple: only the specific Process class (such as ProcessActions) needs to be modified. Here are the steps:

- The name of the widget needs to be added in the table of Strings defining elements of the Combo Box of the ConstructChoice Panel, in *makeSpecComboBox*
- A case has to be added in the switch of *makeSpecOptions* to define options for this Construct
- Finally a case has to be added in the switch of *createSpecConstruct* to generate the HTML file for that construct

6.4.4 Adding a pattern

Adding a pattern requires some more implementation as seen before, but efforts have been done to make it as simple as possible:

- A specific Pattern Class has to be created in the pattern package with methods to handle the ArrayList of pattern elements
- A specific PatternElement Class has to be created in the patternElement package with attributes and getters and setters
- The name of the pattern needs to be added in the table of Strings patterns defined in XMLParser
- Event Handlers of the XMLParser need to detect occurrences of the pattern, stock them in an ArrayList defined in that class and stock elements of this pattern thanks to the specific Pattern object
- The type has to be handled in the generic method *getPattern* of the XMLParser and in the generic methods of ProcessGen
- Last but not the least, a specific Process class needs to implement the Interface ProcessSpec

6.4.5 Implementation effort

As seen through this chapter, this prototype deals with one widget for one pattern currently, but efforts have been done to have a handy tool (integrated web browser, parameters for widgets, User interface, ...) and the addition of new widgets and patterns has been made simple.

A new widget and a new pattern will be added as soon as possible to validate previous subsections.

Conclusion

This chapter will summarize the work done for this MSc thesis, explain its actual contributions, both conceptual and under the form of implementation.

Then it will summarize what else could have been analyzed and how to improve the prototype.

Finally, the last part will be the actual conclusion of this thesis.

7.1 Contributions

Because of the quick advancement done in the field of RIA, learning and defining them precisely, analysing the current frameworks and technologies around them was necessary before anything else.

Once the lack of a model-based approach has been confirmed, an in-depth learning of the MDA was a second step to try addressing this lack.

Then the beautification process and the existence of other technologies to improve code generation such as transformation templates needed to be learned.

Before doing any implementation, understanding technologies and languages around RIA was necessary. This step took quite a long time because those technologies differ from traditional desktop programming technologies. New languages such as (X)HTML, Javascript and the DOM had to be learned and even for a known language such as XML the knowledge has to be improved by learning more precisely its syntax, efficient parsing, Xpath and XSLT transformations. Frameworks such as Aptana, Web Developer toolbar and Oxygen were indispensable. Traditional libraries such as JQuery, YUI but also dojo and script.aculo.us had to be learned too.

Manual implementations allowed to practice all these new concepts and try some scripts.

Understanding the basic use of ONME, its patterns and the generated XML code (where patterns were really hard to identify) was a last step to achieve before having a sufficient knowledge to make significant contributions.

The first contribution providing a real novelty was the analysis of the applicability of RIA constructs for ONME patterns. With instructions from Mr Vanderdonck, the help of both an RIA expert (Francisco Javier Martinez Ruiz) and an expert of ONME patterns (Nathalie Aquino), it was possible to validate this analysis.

The second actual contribution is obviously the beautification editor. Although a prototype, it meets the original objectives of that MSc thesis, i.e. proving that RIA constructs can be used to improve RIA generated by OlivaNova and is designed to be easily improved with new constructs and patterns.

7.2 Improvements

As part of a MSc thesis, both the analysis and the tool are mainly focused on the generation of RIA. They could be extended to handle any kind of application.

The beautification process could be applied to other code generating softwares as well and to other parts of applications than user interfaces.

The beautification is made on patterns without taking their context into account. Taking them into account would be a big challenge.

The tool itself could be improved to handle other constructs and patterns but also to add user interaction (for instance a means of providing images for the Carousel) or on the contrary automate some default parameters (for instance set both the default amount of visible items and the Carousel size depending on the number of items and the space available).

Finally, the third step of the beautification process, i.e. providing a new Concrete User Interface Model to the Model Compiler could be implemented. It would require ONME being able to understand it and generate RIA first.

7.3 Final word

This work contributed to provide more information and a tool as a starting point to achieve both first steps of the beautification process for RIA with ONME.

It could also be reused as a basis for a similar reasoning in other programming environments or with other automated code generation approaches.

The analysis of the applicability of RIA constructs to ONME patterns could also be used as an idea for manual implementation in RIA.

Incidentally it could also serve as a base for discovering RIA and choose the most suited development environment.

Bibliography

- [ABRA08] Silvia Abrahão, Emilio Iborra and Jean Vanderdonckt, *Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool*, 2008
- [ALLA02] Jeremy Allaire, *Macromedia Flash MX - A next-generation rich client*, <http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf>, March 2002
- [BOZZ06] Alessandro Bozzon, Sara Comai, Piero Fraternali, Giovanni Toffetti Carughi, *Conceptual Modeling and Code Generation for Rich Internet Applications*, Dipartimento di Elettronica e Informazione Politecnico di Milano, 2006
- [CARE09] CARE Technologies - OlivaNova The Programming Machine, <http://www.care-t.com/products/index.asp>, last consultation: 27/04/2009
- [FLCA] Adobe, *Flex Application Performance: Tips and Techniques for Improving Flex Server Performance*, http://www.adobe.com/devnet/flex/articles/server_perf_02.html, last consultation: 1/08/09
- [FLPE] Adobe, *Flash Player Penetration*, http://www.adobe.com/products/player_census/flashplayer/, last consultation: 1/08/09
- [FXML] FlashiXML, <http://www.usixml.org/index.php?mod=pages&id=24>, last consultation: 1/08/09
- [GA05] Jesse James Garrett, *Ajax: A New Approach to Web Applications*, 2005, <http://adaptivepath.com/ideas/essays/archives/000385.php>, last consultation: 30/07/09
- [JQ] JQuery Javascript Library, <http://jquery.com/>, last consultation 19/08/09
- [JQUI] JQuery UI Javascript Library, <http://jqueryui.com/>, last consultation 19/08/09
- [JWB] Native Swing Library (with JWebBrowser), <http://djproject.sourceforge.net/ns/>, last consultation: 18/08/09
- [LASZ05] Laszlo Systems Technology White Paper, *OpenLaszlo - An XML Framework for Rich Internet Applications*, Laszlo Systems, 2005
- [MART06] Francisco J. Martínez-Ruiz, Jaime Muñoz Arteaga, Jean Vanderdonckt, Juan M. González-Calleros and Ricardo Mendoza, *A first draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications*, 2006
- [MA07] Francisco Javier Martínez Ruiz, *A Development Method for User Interfaces of Rich Internet Applications*, Université Catholique de Louvain (Diploma of Extended Studies in Management Sciences), 2007
- [MSP] MultiSplitPane: Splitting Without Nesting, <http://today.java.net/pub/a/today/2006/03/23/multi-split-pane.html>, last consultation: 17/08/09

- [NODA05] Tom Noda and Shawn Helwig, *Rich Internet Applications: Technical Comparison and Case Studies of AJAX, Flash, and Java based RIA*, <http://www.uwebc.org/opinionpapers>, 2005
- [OMG09] Object Management Group (OMG), <http://www.omg.org/mda/committed-products.htm>, last consultation: 27/04/2009
- [P&a107] Inés Pederiva, Jean Vanderdonckt, Sergio España, Ignacio Panach and Oscar Pastor, *The Beautification Process in Model-Driven Engineering of User Interfaces*, INTERACT 2007
- [P&M07] Oscar Pastor and Juan Carlos Molina, *MDA in practice: a Software Production Environment Based on Conceptual Modeling*, 2007
- [PREC07] J.C. Preciado, M. Linaje, S. Comai, F. Sánchez-Figueroa, *Designing Rich Internet Applications with Web Engineering Methodologies*, Quercus Software Engineering group. Universidad de Extremadura, 2007
- [SEI07] Software Engineering Institute, http://www.sei.cmu.edu/legacy/case/case_what_is.html, 2007
- [STAT] Statistics on Flash, Silverlight and Java plugin deployments <http://riastats.com/#>, last consultation: 1/08/09
- [WPMDE] http://en.wikipedia.org/wiki/Model-driven_engineering, last consultation: 18/04/09
- [WPRIA] http://en.wikipedia.org/wiki/Rich_internet_applications, last consultation: 26/04/09
- [YUI] The Yahoo! User Interface Library, <http://developer.yahoo.com/yui/>, last consultation 19/08/09