IDEPRIA: A CASE tool for rapid prototyping of Rich Internet Applications

Víctor Manuel Cajes González

Master of Science

Computer Science

School of Informatics

University of Edinburgh

2017

Abstract

Rich Internet Applications (RIAs) are Web applications that provide some features aimed to improve the end-user experience. They can be developed by following Model-Driven Development (MDD) approaches, which regularly provide CASE tools that permit the modelling of these applications and the possibility to transform these models into the ready-to-deploy source code of the RIAs. This source code is typically just a prototype of the desired application, which can be used to verify if the software requirements were met.

Many of the current MDD approaches have different advantages and drawbacks that make them useful for many modelling use cases. However, a comparative analysis presented in this work showed that it is still possible to improve the current approaches. Then, this master thesis developed a Domain-Specific Modelling Language (DSML) aimed for modelling RIAs, the IDEPRIA Web CASE tool that provides support for the defined modelling language, and a Model-To-Text transformation tool that can transform the models into the source code of the modelled RIA.

To evaluate this work, a case-study was executed by following the Think-Aloud protocol to check if CASE tools that have some specific properties could be a well-suited and usable solution for modelling and developing prototypes of RIAs. The results obtained were not conclusive because only three participants were able to test the IDEPRIA tool during the evaluation. Nonetheless, they all learned and used the tool on-the-fly and finished the required tasks in a reasonable amount of time. Thus, promising results were obtained and further research in the field would be appropriate.

Acknowledgements

This work is dedicated to my family because they are always supporting me.

I would have not been able to finish this dissertation without the help and advice of my tutor, Prof. Perdita Stevens.

The possibility of studying in the UK would have not been possible without the funding provided by the *FCO Chevening Scholarship*. I am very grateful to the UK government and the FCO for this amazing opportunity.

To comply with the terms and conditions of the *FCO Chevening Scholarship*, I add the following statement to this dissertation:

"Chevening Scholarships, the UK government's global scholarship programme, funded by the Foreign and Commonwealth Office (FCO) and partner organisations."

Declaration

I declare that I have read and understood the University of Edinburgh's plagiarism guidelines.

I declare that this MSc dissertation was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Víctor Manuel Cajes González*)

Contents

Chapter 1

Introduction

The development of Web 2.0 applications became a trend due to the vast availability of Web frameworks that facilitated the design and implementation of responsive Web interfaces [Jeon and Lee, 2007]. The evolution of the Web browsers and the improvement of these Web frameworks were a combination that simplified the development of *Rich Internet Applications (RIAs)*. RIAs are Web applications that share features that are aimed at improving the whole end-user Web experience [Fraternali et al., 2010]. The most characteristic features of the RIAs were: the ability to avoid reloading the entire web page after each user request; the drag and drop facilities provided by Web browsers; the minimization of the Web server's response time; and multimedia animations [Busch and Koch, 2009].

Over the last ten years, the Model-Driven Development (MDD) approach has been used by software developers in the construction process of Web applications. MDD makes use of models as main artefacts to build the Web applications [Bozzon et al., 2006]. Also, far from trivial modifications and extensions have been made to existing MDD methodologies to allow the modelling of specific RIA elements, including presentation behaviours, distribution of the processing between the client and the server, flexible management of events, and communication [Fraternali et al., 2010]. Then, the complexity of these RIA features became the reason that caused trouble to developers while modelling RIAs.

Fast prototyping is a cost-effective and quick way to produce a minimum viable version of a software system that could be useful for some purpose (e.g.: requirements validation), which can then be transformed and refactored until it reaches the wanted version of the software [Bernardi et al., 2014]. Fast prototypes of RIAs can be developed by converting the models to source code using an MDD approach and in this way developers can discuss these prototypes with other

non-developer stakeholders of the project. Also, a very useful property is that the automatically generated prototype can be used as a starter source code for the final RIA that is wanted.

One of the main disadvantages of the well-known Model-Driven Development OOH4RIA approach to build RIAs is the time that the developers have to invest to learn the approach and the provided tool before they can obtain useful results [Martínez et al., 2013]. The problem could be the multiple models required to develop just a single Web application and the fact that the developer has to understand how these different types of models are related. To support the importance of this problem, we can infer that the OOH4RIA is targetted at single software developers rather than multiple ones per project because the authors mention at least three times that "the designer" is the one that should accomplish a given task [Meliá et al., 2008].

The industry standard language for modelling is the Unified Modeling Language (UML), which has been widely used to model Web Applications [Fowler and Scott, 2000]. One of the many reasons[1] to use UML is because "it's the only software design notation that you can expect your peers to be familiar with", which means that an average software engineer will be able to build and understand a UML model without much trouble. Also, a Web survey revealed that the *UML class diagram* was the one that provides more new information (when compared to other UML diagrams) and the most frequently used among many developers [Dobing and Parsons, 2006].

UML class diagrams have a characteristic that allows the automatic derivation of an *object-relational schema* [Golobisky and Vecchietti, 2005], and thus, this feature can be exploited by M2T transformation tools to generate the source code of the Web application that is aimed at storing persistently instances of the each of the possible UML classes available in a given *UML class diagram*. From this diagram, it is feasible for an M2T transformation tool to generate the four basic functions of persistent storage (CRUD): create, read, update, delete [Albert et al., 2011]. These four functions could be implemented as *Web services*[2] in the back-end to provide support to the front-end of the RIA. Also, as the goal is to obtain just a RIA prototype as fast as possible, then much customization of the RIA front-end is not required, and thus, the developer will make a better use of the time by modelling important aspects of the system rather than modelling look and feel aspects of the RIA front-end.

To summarize, given a *UML class diagram*, it is possible to use an M2T transformation tool to automatically generate the source code of the Web services that are implemented in a back-end application to provide the four CRUD functions, a RIA front-end application by using RIA

[1] Reason to use and do not use UML: `https://saturnnetwork.wordpress.com/2010/10/22/five-reasons-developers-dont-use-uml-and-six-reasons-to-use-it/`

[2] `https://www.w3.org/standards/webofservices/`

frameworks to provide a Web graphical user interface, and the data access objects that will make possible to persistently store instances of the models by using a database engine. Then, the combination of the front-end, database schema, and back-end together are the desired RIA prototype.

Given the previous facts and ideas, this master project is about designing and developing a CASE tool to allow software developers model RIAs by requiring only one familiar extended *UML class diagram*, so then an M2T transformation software can transform those models into a ready-to-deploy source code of the modelled software. Also, the tool is focused on *rapid prototyping*, which means that this CASE tool will be able to generate a testable version of the modelled software so it can be used by the stakeholders of the project for discussing detailed aspects of the requirements.

The goal of this master thesis is to provide a Web CASE tool that: removes the requirement of having to install one or more software components to start modelling by developing the CASE tool as a Web application (which just requires a Web browser); solves the problem of having developers spending much time learning how to model the different type of required models in current available MDD approaches and then learning how to link the elements between them; that provides specific design elements aimed to simplify the modelling of prototypes of RIAs by requiring just one extended and probably familiar *UML Class Diagram*.

Finally, the hypothesis is that: "A CASE tool that just requires one familiar diagram could be a well-suited and usable solution for developing rapid prototypes of Rich Internet Applications". A case-study was performed to analyse this hypothesis, which resulted in promising results but without being able to confirm or proof the statement.

This dissertation is structured in the following way: Chapter 2 provides a detailed background of the RIA and MDD concepts that the reader show know to understand the solution the problem that this master thesis is trying to solve. Chapter 3 shortly describes the current approaches available in the literature, what are their advantages and drawbacks, and why a new proposal could still contribute to the state of the art of MDD approaches aimed to simplify the development of RIAs. Chapter 4 describes in great detail the three elements designed and implemented as a solution: the Domain-Specific Modelling Language, the IDEPRIA Web CASE tool, and the M2T transformation software. Chapter 5 shows the design, execution and results obtained in the case-study. Finally, the last chapter provides concluding remarks, additional observations, unsolved problems, and further work that could be done in the area.

Chapter 2

Background

This chapter provides a summary of the theory related to the aforementioned topics in the Introduction section. The first section describes the Rich Internet Applications (RIAs) to show to the reader why are they important when developing Web applications.

The second section summarises the Model-Driven Development (MDD) approach and what are its main advantages and disadvantages. Also, the importance of UML is emphasized and how it can be extended to define Domain-Specific Modelling Languages (DSMLs) that could be used as part of a MDD approach.

2.1 Rich Internet Applications (RIAs)

Rich Internet Applications are defined as "web applications, which use data that can be processed both by the server and the client. Furthermore, the data exchange takes place in an asynchronous way so that the client stays responsive while continuously recalculating or updating parts of the user interface." [Busch and Koch, 2009]. Moreover, Rich Internet Applications refers to a set of ideas and frameworks that intend to add new features to the hypertext-based Web to improve the whole user Web experience [Fraternali et al., 2010].

In the previous Web 1.0, the Web clients (e.g.: Web browsers) just requested data from Web servers when needed, which were the ones that hold all the data. On the other hand, RIAs can store information in the Web clients [Fraternali et al., 2010], which could allow RIAs to run in offline mode without even having a connection to the Web server. For instance, the *W3Schools*

website provides a tutorial on "How to use the Web local storage"[1], which shows how developers can build Web applications that use Web local storage. Additionally, a feature of the RIAs is that they enable moving part of the computation from the server to the client [Fraternali et al., 2010], and this is very important because:

1. the web servers will have more available resources to serve more request or perform other tasks because the clients now can perform part of the computation.
2. there will be a reduced latency in the clients because now they can perform some tasks without asking a Web server. For instance, sorting a table by a given column without requesting anything to the server.
3. fewer requests from the Web clients to the Web servers implies less overall bandwidth consumption.

RIAs bring the possibility to Web servers to send a message to a client without having to wait for a Web client request [Fraternali et al., 2010]. This is a way to save bandwidth and computing resources of Web servers because it provides a solution to the *pull requests*. For instance, a given server could have a bad performance if many Web clients start making pull requests to check for some updates (even if there are no updates), instead of just having the server pushing updates individually to clients interested in receiving such updates.

Another useful capability of the RIAs is their flexibility, which allows a single web page to have multiple sub-pages that can handle user events rather than having the Web server handling user events and requiring the reloading of the entire web page to display a little change [Fraternali et al., 2010]. This improves the user experience because the user does not have to wait for the Web server's response. However, this might incur in an extra complexity for the RIA font-end developers because the Web client:

1. should handle local events received from the Web graphical user interface.
2. needs to parse the message received to execute the appropriate callback.
3. have to apply changes to the GUI to show the new system status to the user.

Figure 2.1 shows an example[2] of a RIA that makes use of many of the features previously mentioned. At the bottom left of the image, you can see a start menu that only uses client-side computing to display a menu with many possible actions available to the user without contacting

[1] W3Schoools local storage: `http://www.w3schools.com/html/html5_webstorage.asp`

[2] RIA Example: `http://docs.sencha.com/extjs/4.2.4/extjs-build/examples/desktop/desktop.html`

the server. Also, at the centre of the image, you can see a table that displays information that was loaded from the back-end but without having to request from the server the entire web page.

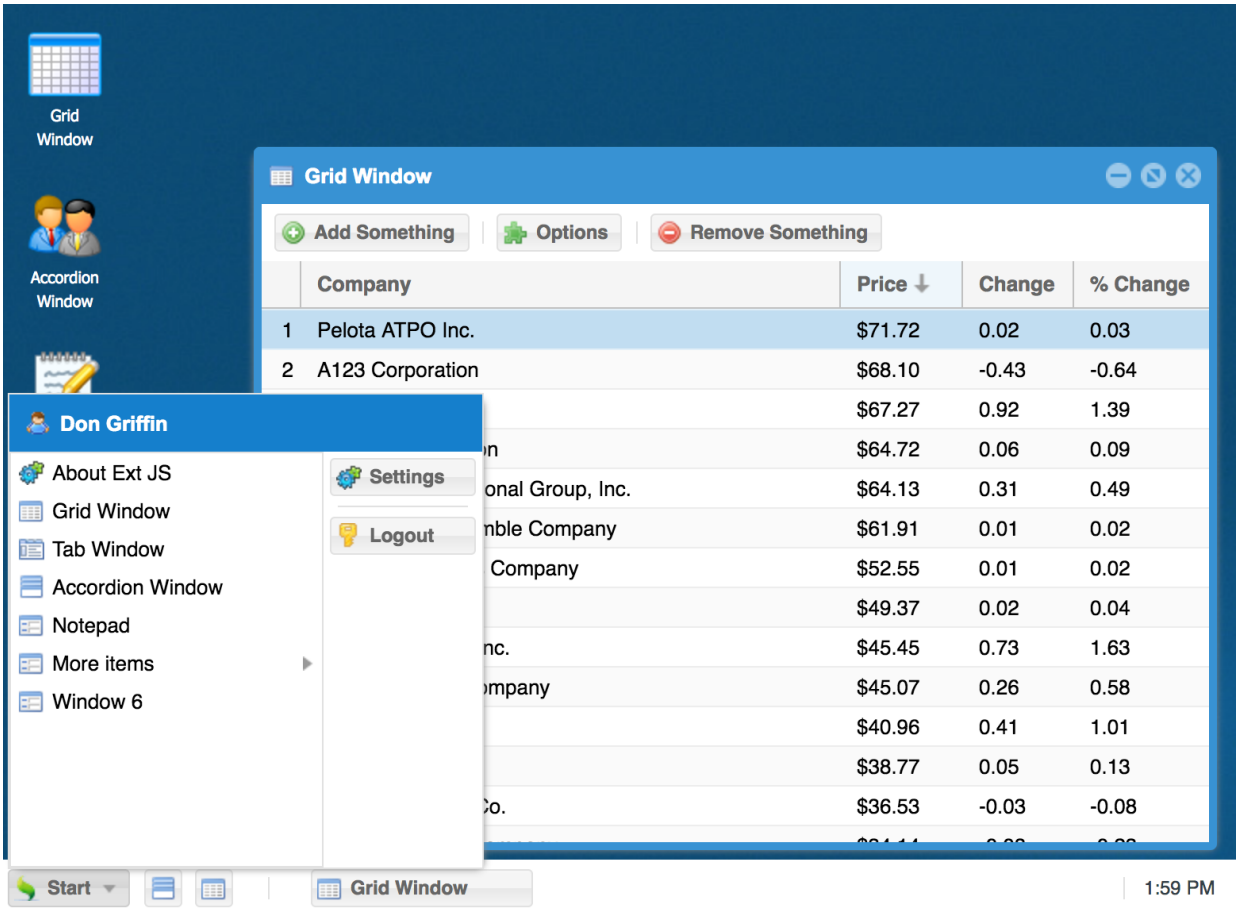

Figure 2.1: Example of a Rich Internet Application.

2.2 Model-Driven Development (MDD)

A *model* is "an abstract representation of a specification, a design or a system, from a particular point of view" [Stevens, 2006]. One way to define models is by using a *modelling language*, which is a graphical way of representing the various models built while designing a software system [Stevens, 2006]. A modelling language normally provides a set of elements (word, sentences, graphics) that are used to define the *syntax* (which establishes if a given model is valid or not) and the *semantics* (the meaning of a valid model) [Stevens, 2006].

Model-Driven Development is an industry term for round-trip engineering of working software source code using CASE modelling tools [Ambler and Lines, 2012, p. 328]. This is a software development approach, in which the models are the core artefacts of the software being developed

because they are then transformed into the source code of the modelled software system with the help of a *Model-to-Text* transformation software [Marco Brambilla, 2017].

One important reason to choose the Model-Driven Development approach instead of code-centric approach for developing a given software system is the productivity improvement that could be achieved [Selic, 2003]. There are case studies that show that MDD approaches allowed developer teams to finish 2.6 [Krogmann and Becker, 2007], 8.0 [Kapteijns et al., 2009], and 10.1 [Martínez et al., 2012] times faster than the teams that developed the same software following a code-centric approach. However, [Selic, 2003] emphasises the fact that if an erroneous state of the generated source code is detected, then it is difficult to trace the error and find the exact component of the model that contains the error, which could add extra delays to the project.

Additionally, one useful feature of the MDD approach is that the model artefacts obtaining during the modelling phase are related much closer to the problem domain rather than the underlying implementation of the software wanted [Selic, 2003]. This is because developers can get a ready-to-deploy source code of the application that could be written in many different programming languages, that use a wide variety of frameworks, and even with the ability to work with various database engines. Nevertheless, to achieve this, developers first have to learn how to make syntactically and semantically correct models that can then be transformed into source code.

The *Object Management Group (OMG)*[3] is an open membership, international, and nonprofit technology standards consortium, which put a lot of effort into designing and developing the standard modelling language: *The Unified Modeling Language (UML)*[4]. This language is very important for the Model-Driven Development because "UML is a language for constructing", which means that the model forward engineering technique can be applied to UML models to transform models into the source code of the modelled software [Booch, 2005].

Domain-Specific Modelling Languages (DSMLs) are graphical languages that address the needs of specific software domains [Marco Brambilla, 2017]. These languages do not force to the users to study General-Purpose Modelling Languages (GPML) that might contain irrelevant elements for the domain of study, while at the same time DSMLs provide proper modelling abstractions that are very related to the domain [Marco Brambilla, 2017].

There are mainly two ways of designing and defining a DSML [Marco Brambilla, 2017]:

1. **Defining an entirely new DSML**: requires the definition of the abstract syntax (the structure of the language), the concrete syntax (specific representations of the modelling com-

[3] Object Management Group (OMG): `http://www.omg.org/`
[4] UML specifications: `http://www.omg.org/spec/UML/`

ponents) and the semantics (the meaning of the elements) [Marco Brambilla, 2017]. For example, *MetaEdit+ Modeler*[5] permits the definition of the modelling language, and then the usage of the modelling language to build domain specific models.

2. **Extending an existing General-Purpose (GPML)**: simpler than defining a whole new DSML because elements of the GPLM can be reused [Marco Brambilla, 2017]. Also, UML provides extensibility features through stereotypes, constraints, tagged values and profiles [Marco Brambilla, 2017].

The UML extension that provides support for a specific domain is defined in the *UML Profile Diagram*, which defines the packages of related and coherent extensibility elements (also known as UML Profiles) [Marco Brambilla, 2017]. The *stereotypes* were defined in UML to extend meta-classes to add additional semantics to the meta-class concept [Marco Brambilla, 2017]. A stereotype can be defined by specifying this properties [Marco Brambilla, 2017]:

- *Base meta-class*: specifies the element that is going to be extended. For instance, the UML meta-class **Class** can be extended to define a stereotype named **WebTable**, which then a developer will be able to use it in a given modelling project to represent a real Web table that would show information.
- *Constraints*: particular rules and semantics that apply to a given stereotyped element.
- *Tagged values*: tag-value pair that might be attached to a stereotype. These values are very useful to add extra information to an element, that can be used later by the M2T software to generate a source code with a specific behaviour.
- *Icon*: the visual appearance of the stereotyped element. This icon is very useful because allow developers to recognize modelling elements by just looking at the icon of the element without having to access the details of that element.

Figure 2.2 shows an example of a UML Profile taken from [Marco Brambilla, 2017] that allows the modelling of *Enterprise Java Beans (EJB)* concepts. In the example, the defined stereotype *Bean* extends the UML meta-class *Component*, and the other stereotypes, *Entity* and *Session*, extend from the stereotype *Bean*. The tagged value *state* was attached to the *Session* stereotype, and this possible values that this *state* tag could take are defined in the *StateKind* data type enumeration: *stateless* and *stateful*. In addition, the reader can find another example of how to define and use UML Profiles in [Fuentes and Vallecillo, 2004].

For Model-Driven Development purposes, once the Domain-Specific Modelling Language is de-

[5] MetaEdita+ Modeler: `http://www.metacase.com/mep/`

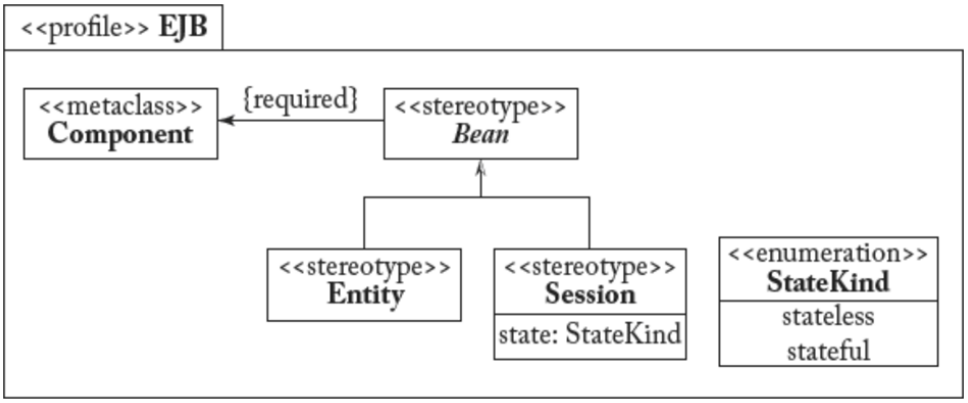

Figure 2.2: Example of a UML Profile. Image source: [Marco Brambilla, 2017]

fined, the appropriate semantics should be defined so each modelling element can have a well-defined meaning, which then will be used by a *model-to-text* transformation software to transform the models into the source code of the modelled application.

Nowadays, there are mainly two ways of performing code generation, through programming languages, and through M2T transformation languages [Marco Brambilla, 2017]. The code generation approach through programming languages is simple and can be achieved by writing a program in a general-purpose programming language that interrogates the model and writes out generated code [Marco Brambilla, 2017]. Nevertheless, this approach has several drawbacks:

- "intermingled static/dynamic code": there is no separation and differentiation of static code (e.g.: code that does not depend on any model and is generated always exactly), and dynamic code (e.g.: code generated for a custom class of the model).
- "non-graspable output structure": the output structure is embedded into the code generator that contains as well static/dynamic code. Then, it is hard to visualise the output locations of the static/dynamic generated source code files.
- "missing declarative query language": this leads to many iterators, loops, type casts and conditions that lead to unnecessarily huge amounts of lines of code in the generator.
- "missing reusable base functionality": every time that a new output is desired, the source code of the generator needs to be changed.

To eliminate the previously mentioned disadvantages, *transformation languages* have been developed for generating text from models [Marco Brambilla, 2017]. This approach separates static and dynamic code by using templates, which are a blueprint that is made of dynamic parts that will be filled with information from the models and static text shared by all artefacts [Marco Bram-

billa, 2017]. Thus, the templates contain static text and placeholders (meta-markers) that are going to be interpreted by the template engine to produce the desired output [Marco Brambilla, 2017]. Also, the templates allow the explicit representation of the structure of the output and the use of a declarative query language by using meta-markers, which are aimed to query information stored in the models [Marco Brambilla, 2017]. Finally, one of the most important advantages of this approach is the "reusable base functionality", which allows the generation of multiple possible outputs by just using configuration files that guide the M2T translation process, without having to change the source code of the generator [Marco Brambilla, 2017].

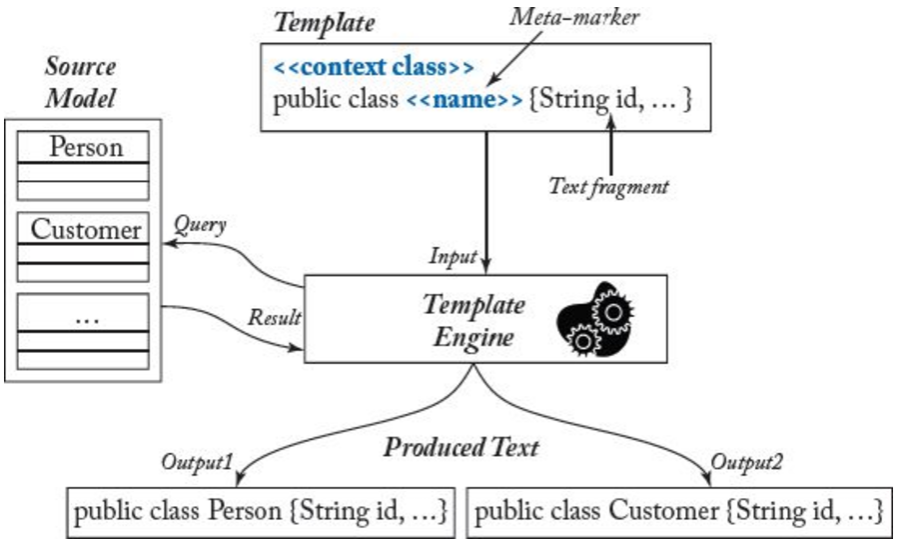

Figure 2.3: Template-based M2T transformation. Image source: [Marco Brambilla, 2017]

Figure 2.3, which was taken from [Marco Brambilla, 2017], shows the template-based M2T transformation process that begins with input source models and predefined templates that are fed to the template engine, which then generates the appropriate output source code. The same template can be used for many models and vice-versa to obtain the desired output.

To summarise all the concepts discusses in this section, UML is the standard language for modelling, which can be extended by defining UML Profiles. These UML Profiles define model elements that could be used for the domain of the Rich Internet Applications. Then, developers can use these UML extensions to model specific Rich Internet Applications, which then can be transformed into the source code of the application by using a M2T transformation software.

Chapter 3

State of the art

This chapter analyses multiple Model-Driven Development approaches aimed at developing Rich Internet Applications. The first section describes the criteria that could be used to evaluate these approaches and also explains why these criteria are meaningful for the goal of this master thesis.

The second section lists many of the MDD approaches for RIAs and then describes the comparative analysis performed to understand the advantages and disadvantages of these approaches. Finally, the third section shows the results of the comparative analysis performed between the MDD approaches, and presents a short conclusion.

3.1 Criteria to evaluate Model-Driven Development approaches

The criteria that could be used to evaluate MDD approaches for RIAs and their importance are:

- *Licenses and pricing*: some MDD approaches would just provide a free CASE tool like [Meliá et al., 2008], which uses the *Eclipse Modeling Framework (EMF)*[1]. However, there is another approach called *Appcelerator Titanium*[2], which requires payments every month to obtain access to the tool. Then, the cost of the tool may be a problem that could restrict the access to a specific MDD approach.
- *Usability of the CASE tool provided*: MDD approaches very often provide tools that should be used to model a specific software. These tools might hamper the entire development

[1] Eclipse Modeling Framework: `https://www.eclipse.org/modeling/emf/`
[2] Appcelerator Titanium Pricing: `https://www.appcelerator.com/pricing/`

process if they are not usable, and thus, developers might just decide to not follow a given approach for being too complex and difficult to obtain something working.

- *Use of UML diagrams*: As UML is the standard language for modelling, then MDD approaches that define their own language will probably incur a longer learning curve because developers will have to first learn the language, and then learn how to make specific models with that language.

- *Number of types of models required*: a significant number of required models would probably mean that more details of the given application can be modelled and described. For example, [Bernardi et al., 2014] does not provide a way to model the user behaviour within any of the three possible models that should be made with that approach, in contrast to the [Moreno et al., 2008] that allows modelling the user behaviour in one of the six possible models that should be built. Nevertheless, the downside of having many models includes the fact that developers will have to understand all those models and how they should be linked to each other to model the specific application, and that might also introduce extra delays in the process.

- *Availability of Web Modelling Tools*: the already mentioned Web trend removes the need of installing software in the computer by allowing to run applications in Web browsers [Breeding, 2012]. Then, there are MDD tools that can be used entirely in Web browsers, like *AppFlower*[3]. However, there are as well approaches that require the Eclipse Modelling Framework software to be installed locally on the computers that are going to be used to model the software system [Meliá et al., 2008].

- *Platforms of the generated applications*: RIAs can be developed with many different client libraries (e.g.: Dojo, OpenRico) and server libraries (e.g.: Python Django[4], Java Spring-MVC[5]) [Fraternali et al., 2010]. Then, it would be a great feature for a given MDD approach to have the ability to generate Rich Internet Applications that could be made of different combinations of programming languages, back-end and front-end frameworks, and support to multiple database engines. This is because, for instance, developers of a given company might have experience in Python but not in Java, then, they might not choose a given MDD approach because the RIA that is going to be generated from the models will necessarily be made of source codes written in Java (and they do not know Java).

- *Extensible template-based M2T transformations*: as described in the previous chapter of

[3] AppFlower: `http://www.appflower.com/`
[4] Python Django Framework: `https://www.djangoproject.com/`
[5] Java Spring Framework: `https://projects.spring.io/spring-framework/`

this dissertation, [Marco Brambilla, 2017] explains the reasons why template-based M2T transformations are better than programming languages based M2T transformations. To remember, the benefits of the template-based M2T transformations include the separation of static and dynamic code, the explicit definition of the output structure, the features of the declarative query languages, and the reusable base functionality [Marco Brambilla, 2017]. These criteria evaluate if a given approach follows or not the recommended M2T transformation technique.

3.2 Current projects for building RIAs

The idea of having a CASE tool that allows the modelling and generation of prototypes of RIAs is not new but is important to show to the reader the advantages and limitations of the current approaches in order to understand the motivation of this master thesis.

Nowadays, there are many projects that allow developers to build RIAs. A well-maintained list of projects can be found in this Wikipedia entry on "RIA frameworks"[6]. Due to the fact that there are many of these projects, to perform the comparative analysis of these projects, a two-stage protocol has been defined in the following way:

- **First stage**: in this stage, all the projects that do not provide a CASE tool that could allow the modelling and generation of prototypes of RIAs will be discarded.
- **Second stage**: in this stage, the remaining projects will be compared using the criteria mentioned in the previous section: licenses, costs, usage or not of UML, the number of types of models required, availability of Web modelling tools, possibles output platforms, and the extensibility through template-based M2T transformations. Also, I did not make any usability evaluation on the projects because of the amount of time that doing it properly would have taken.

To determine whether a given project satisfies a given criterion, I used the information (descriptions, images, screen captures, descriptive videos, tutorial videos) provided on their respective official websites.

To begin, the alphabetically sorted list of projects to analyse and a brief description of them are:

- **Adobe Flex**: software development kit that allows the development and deployment of

[6] RIA frameworks: `http://en.wikipedia.org/wiki/List_of_rich_Internet_application_frameworks`

RIAs that use Adobe Flash.
Website: `http://www.adobe.com/products/flex.html`

- **AmpleSDK**: JavaScript framework aimed to simplify the development of multi-platforms Web applications.
  Website: `http://www.amplesdk.com/`
- **AngularJS**: JavaScript framework maintained by Google developers that simplify development of RIAs and even mobile Web applications.
  Website: `https://angularjs.org/`
- **Appcelerator**: platform that provides a CASE that allows developers to design and build Web applications that can run on desktop and mobile devices.
  Website: `http://www.appcelerator.com/`
- **Cappuccino**: open-source framework designed to simplify the development of Web applications that should look similar to OSX desktop applications.
  Website: `http://www.cappuccino-project.org/`
- **Dojo Toolkit**: framework for building multi-platform Web applications.
  Website: `http://dojotoolkit.org/`
- **eXtensible Enterprise Objects (XEO)**: framework and CASE tool aimed to provide support to the Model-Driven Architecture (MDA) methodology.
  Website: `http://www.xeoframework.org/`
- **ExtJS**: JavaScript framework that simplify the development of RIAs.
  Website: `http://www.sencha.com/products/extjs/`
- **Google Web Toolkit**: collection of open-source frameworks that allow Web developers build complex Web applications in Java.
  Website: `http://www.gwtproject.org/`
- **JavaFX**: Java framework that allows the development of RIAs in the Java programming language that can run on multiples different platforms.
  Website: `http://javafx.com/`
- **OOH4RIA CASE Tool**: CASE tool aimed at the design and generation of RIAs.
  Website: `http://suma2.dlsi.ua.es/ooh4ria`
- **OpenLaszlo**: open-source platform that allow the development of RIAs.
  Website: `http://www.openlaszlo.org/`

- **PhoneGap**: framework that simplifies the development of mobile applications that follow the recommendations of the W3C.
  Website: `http://www.phonegap.com/`
- **Qooxdoo**: framework that allows the development of complex Web GUIs.
  Website: `http://qooxdoo.org/`
- **SproutCore**: open-source JavaScript framework. Its goal is to simplify the development of Web applications that look similar to traditional desktop applications.
  Website: `http://www.sproutcore.com/`
- **Vaadin**: open-source framework that allow the development of Java RIAs.
  Website: `https://vaadin.com/`
- **wCMF**: open-source framework and CASE that allow the design and model of object-oriented Web applications.
  Website: `http://wcmf.sourceforge.net/`

3.3 Comparative analysis of current projects for building RIAs

3.3.1 First stage analysis

In this stage, the projects that do not provide a CASE tool that allow the modelling of prototypes of RIAs will be discarded. After retrieving information from the project's respective websites, the discarded projects are Adobe Flex, AmpleSDK, AngularJS, Dojo Toolkit, ExtJS, Google Web Toolkit, JavaFX, OpenLaszlo, PhoneGap, Qooxdoo, SproutCore, Vaadin.

The remaining projects that can proceed to the second stage of analysis are Appcelerator, Cappuccino, OOH4RIA, XEO, wCMF.

3.3.2 Second stage analysis

The Table 3.3, Table 3.2, and the Table 3.1 show a summary of the information retrieved from the five projects that provide a CASE tool that allows developers to design and build RIAs.

Criteria	**OOH4RIA**
License	Apache 2.0
Pricing	Free
Use of UML	Use of UML models.
Output platforms	Google Web Toolkit (GWT) Java back-end and GWT JavaScript front-end.
Web CASE tool	No Web CASE provided. An Eclipse-based tool is provided.
Types of models required	Do not require models. A drag-and-drop interface is provided to create HTML elements.
Template-based M2T	Yes.

Table 3.1: Comparison of projects to build RIAs - Part 1

Criteria	**XEO**	**wCMF**
License	Free: GPL v3.0, and Paid private version.	LGPL v2.0
Pricing	Free restricted version. Paid (unknown amount) full version.	Free
Use of UML	Do not use UML. Use a custom notation.	Use of UML models.
Output platforms	Java EE back-end server, ExtJS front-end client	PHP back-end server and basic HTML front-end client.
Web CASE tool	No Web CASE provided. An Eclipse-based tool is provided.	Web CASE is provided.
Types of models required	Entity model, Logic model, and View model.	Entity model, View model and Access Control model.
Template-based M2T	No.	No.

Table 3.2: Comparison of projects to build RIAs - Part 2

Criteria	**Appcelerator**	**Cappuccino**
License	Private	LGPL v2.1
Pricing	99 USD per month	Free
Use of UML	Do not use UML or any custom model notation.	Do not use UML or any custom model notation.
Output platforms	JavaScript-based applications for Windows, Android, iOS	PHP-Python-Ruby back-end servers and Objective-J front-end client.
Web CASE tool	No Web CASE provided. An Eclipse-based tool is provided.	No Web CASE provided. A XCode plugin is provided to edit the HTML view.
Types of models required	Do not require models. A starter source code is provided with drag-and-drop support to add new HTML components.	Do not require models. A drag-and-drop interface is provided to create HTML elements.
Template-based M2T	No.	Yes.

Table 3.3: Comparison of projects to build RIAs - Part 3

Given the previous comparative tables, I am going to emphasise two important conclusions:

1. **No project meets all criteria.**
2. **No project was intended for one UML class diagram only**: the main goal of this master thesis is to evaluate the usability of a CASE tool that just requires one single and familiar UML class diagram to obtain a RIA prototype.

To conclude, with a simple comparative analysis, I identified that the usability evaluation that this master thesis wants to perform still has a chance to contribute to the state-of-the-art in the field of CASE tool for designing and building RIAs.

Chapter 4

IDEPRIA: Integrated development environment for rapid prototyping of RIAs

This chapter describes all the core components of the CASE tool developed for this master thesis, the *Integrated Development Environment for Prototypes of Rich Internet Applications (IDEPRIA)*.

The first section shows the Domain-Specific Modelling Language (DSML) defined by extending UML to permit the modelling of the domain of the Rich Internet Applications. The detailed description of each of the defined design elements can be found in Appendix A.

The second section briefly describes the most time-consuming part of this master thesis, which was the design and development of the CASE tool that allows to developers modelling Rich Internet Applications.

Finally, the third section describes the Model-to-Text (M2T) transformation software that was integrated into the IDEPRIA CASE tool. This is the software that parses and reads a given modelling project and then transforms it to a RIA source code.

4.1 The DSML supported by IDEPRIA

Remembering from the Chapter 2, Domain-Specific Modelling Languages are graphical languages that provide elements that permit the design and development of specific domains, which in this case are the domain of the Rich Internet Applications.

There are mainly two ways of designing and defining a Domain-Specific Modelling Language,

by defining a completely new Domain-Specific Modelling Language or by extending an existing General-Purpose Modelling Language (refer to Chapter 2 for background information). For this master thesis, the final decision was to define the IDEPRIA Domain-Specific Modelling Language as an extension of the Unified Modelling Language (UML). The reasons that justify this decision are:

- The use of tools like *MetaEdit+ Modeler*, which allows the definition of the DSML for the RIA domain using a custom notation and then the possibility to use the same *MetaEdit+ Modeler* tool to design RIAs, are not appropriate for this master thesis. This is because the goal of this project is to evaluate if the developed IDEPRIA Web CASE tool can be a well-suited MDD solution for developing prototypes of RIAs, and there is no point in having defined the DSML with a tool that restricts the usage of the DSML to the same tool. In other words, it is not possible for the IDEPRIA tool to use the language defined in MetaEdit+ (or any other CASE tool that behaves like this one).
- UML is the standard for modelling, and its notation is expected to be known by an average software engineer. Extending the UML implies that at least the common UML elements will still be present in the DSML that is wanted for modelling RIA elements. Then, most developers would probably not have to learn a whole new custom notation for modelling RIAs.
- Finally, there is no need to reinvent the wheel by defining a new custom notation that would probably have again many of the common elements that are available in a General-Purpose Modelling Language like UML. These common elements are for instance: classes and modules, relations and associations, members and attributes, among others.

Given the previous decision, it is important to mention that there are many free/paid tools that can be used to define UML Profiles in order to extend UML to allows modelling of specific domains. Some of these tools are *Eclipse Papyrus*, *MagicDraw UML*, *StarUML*, among others. Also, as all that is needed at this stage is just a formalization of the modelling language that will be supported by the IDEPRIA tool, it does not matter which tool is used to define the DSML. Thus, I just chose Eclipse Papyrus because it is freely available and it is more than enough to define a UML extension for the domain of the Rich Internet Applications.

To define the UML Profile for IDEPRIA with the Eclipse Papyrus tool, I followed the online tutorial that is published on the Eclipse Papyrus website[1]. Due to the dimensions of the paper of

[1] Papyrus Tutorial for UML Profiles: `http://www.eclipse.org/papyrus/resources/PapyrusUserGuideSeries_AboutUMLProfile_v1.0.0_d20120606.pdf`

this dissertation, the UML Profile was split into multiple parts so the images can fit properly. Also, this summary provides just a brief summarised description of the UML Profile to avoid forcing the reader to read the detailed description of all the defined modelling elements (stereotypes, tagged values, associations). The reader should refer to Appendix A to access to the detailed description and justification of each of the modelling elements that are going to be shown in this section.

To begin, Figure 4.1 shows the definition of the stereotype *Project*, which extends from the UML meta-class *Package*. This *Project* stereotype represents a single modelling diagram to which could be added modelling elements (e.g.: classes, attributes, associations) to design a RIA. Appendix A.1 explains all the tagged values shown in this figure.

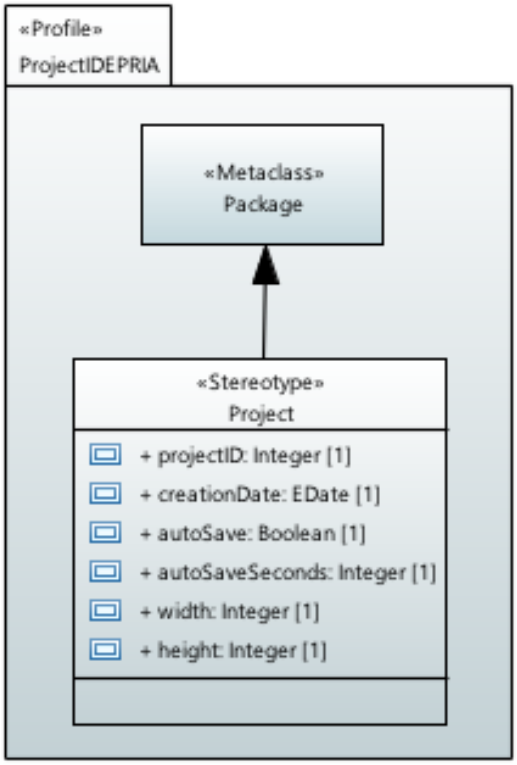

Figure 4.1: UML Profile - Package

Figure 4.2 shows the definition of the core stereotype *NormalClass*, which extends from *ClassIDEPRIA* in order to provide general purpose functionality to RIA elements. There is another stereotype that extends from *ClassIDEPRIA*, the *VideoClass* stereotype, which represents a Web video player that could be added to any given diagram. Also, there are eight required stereotypes (*User*, *UserProfile*, *Profile*, *Action*, *ProfileAction*, *Person*, *Shortcut*, *Module*) that extend from *ReservedClasses* and provide basic features that are normally given in frameworks that allow the construction of RIAs. For example, the website of the framework *Django (Python)* on "User authentication"[2] explains how that framework could be used to give a way to the users to authenticate to the system. Additionally, the website of the framework *Spring (Java)* on "Spring Security"[3] shows the required steps to authenticate users on a given *Spring* Web project. Sometimes, these basic features might not be required in RIAs, but it could be very useful to have them ready to be used in case of needing them in the future without having to change the source code of the application. Appendix A.2 shows the importance and detailed description of the stereotypes and tagged values shown in this figure.

[2] Django authentication: `https://docs.djangoproject.com/en/1.11/topics/auth/`
[3] Spring security: `https://spring.io/guides/gs/securing-web/`

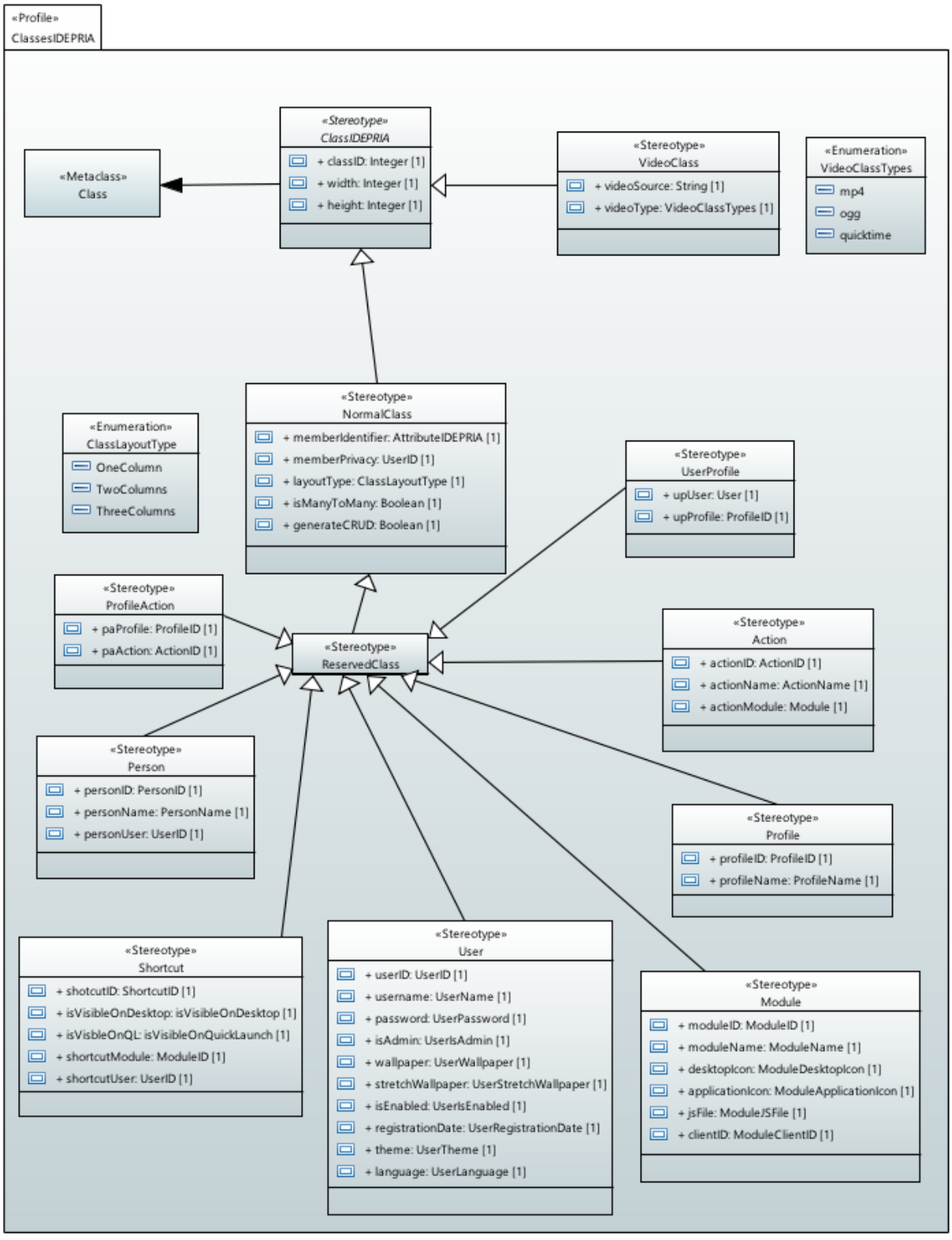

Figure 4.2: UML Profile - Classes

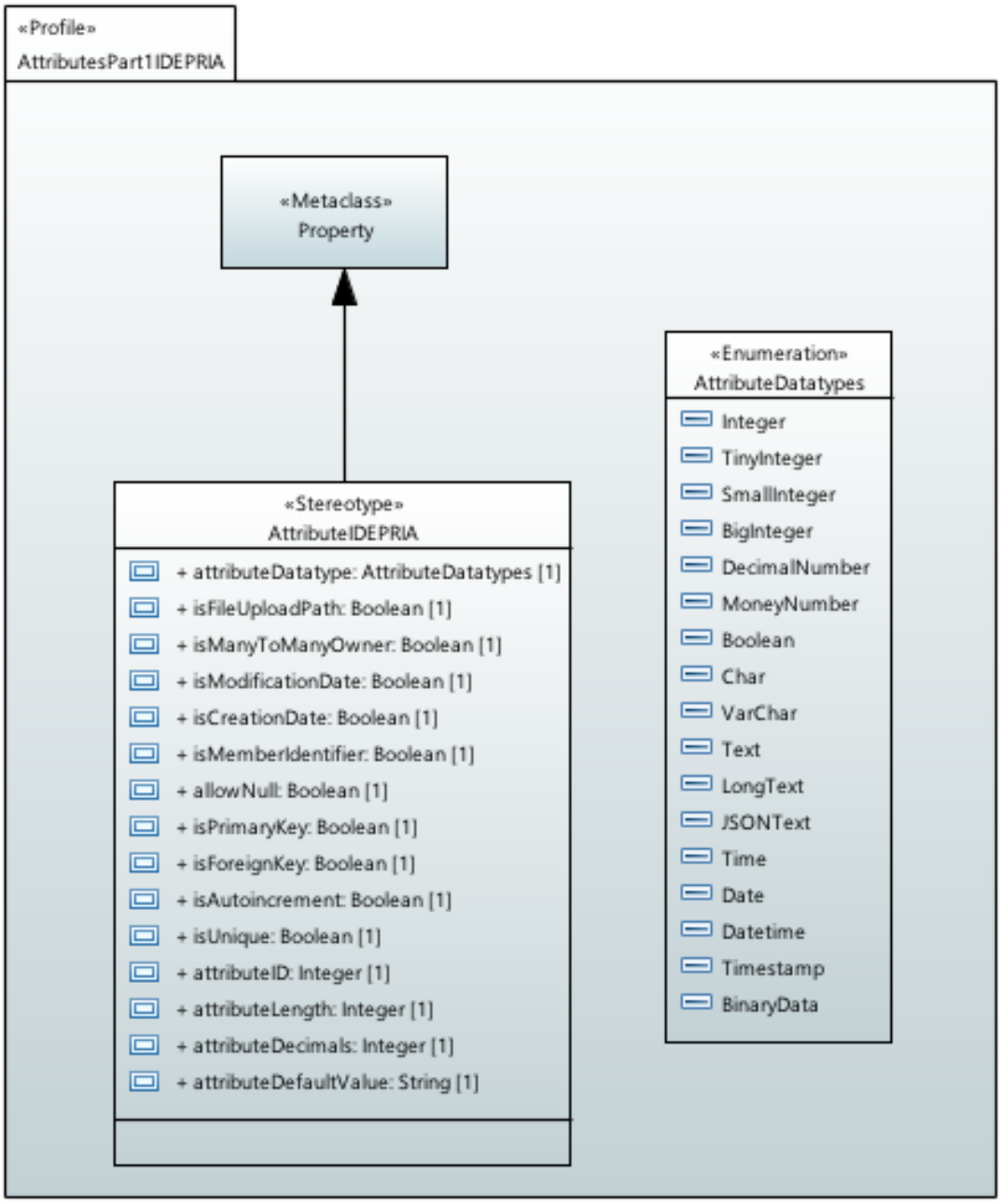

Figure 4.3: UML Profile - Attributes - Part 1

Furthermore, extensions from the UML *Property* meta-class have been defined to provide specific semantics to attributes of classes. Firstly, Figure 4.3 shows the definition of the stereotype *AttributeIDEPRIA*, to which could be attached tagged values that allows a developer to customise basic aspects of any application. For instance, the *allowNull* tagged value is a boolean that determines if a given attribute instance in a modelling project could have ever have or not a null value. This is very important because, for example, the database schema that is going to be generated

for this RIA should define if these specific attribute can have null values or not. The detailed description and the importance of these tagged values can be found on Appendix A.3.

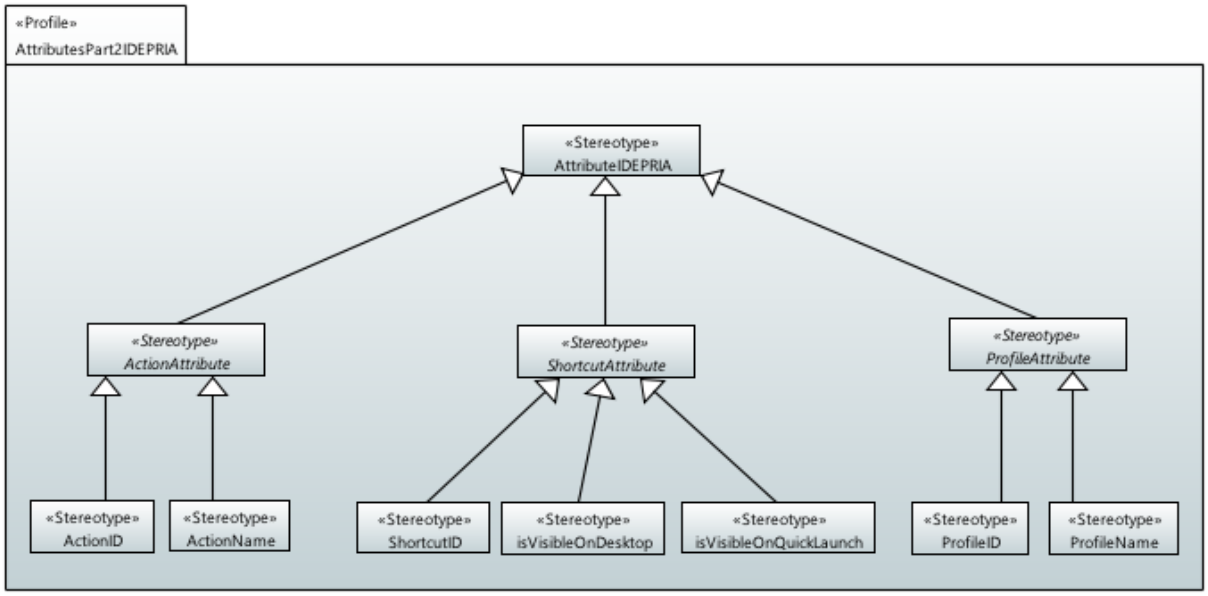

Figure 4.4: UML Profile - Attributes - Part 2

Figures 4.4, 4.5, 4.6 define extensions to the stereotype *AttributeIDEPRIA*, which was shown in Figure 4.3. Mostly, these stereotypes are the ones used as tagged values attached to the stereotypes that extend from the *ReservedClass* stereotype. They are very useful because they determine specific behaviours of RIAs, for instance, the *UserPassword* stereotype that extends from the *UserAttribute* is very important because it tells to the M2T transformation software that this attribute should be encrypted before it gets saved to the database and also that this field should match the password of a given user that is trying to log-in to the RIA. Appendix A.2 provides a description and justification of all the required stereotypes shown in these figures.

Stereotypes can be defined with a specific icon, which might be useful for developers to recognise that a given attribute is configured by just looking at the icon of the modelling element. Figure 4.7 shows an example of a *ClassIDEPRIA* created with the IDEPRIA tool. The stereotypes that have customised icons are:

- *ClassIDEPRIA*: this stereotype is graphically represented with the empty white square, that has a green header with the name of the instance on it. In the example, the “Files” white box with green header and rounded buttons in the four corners that allow end-users to resize the box.
- *AttributeIDEPRIA*: the icon is a green arrow pointing towards the right. In the example, the

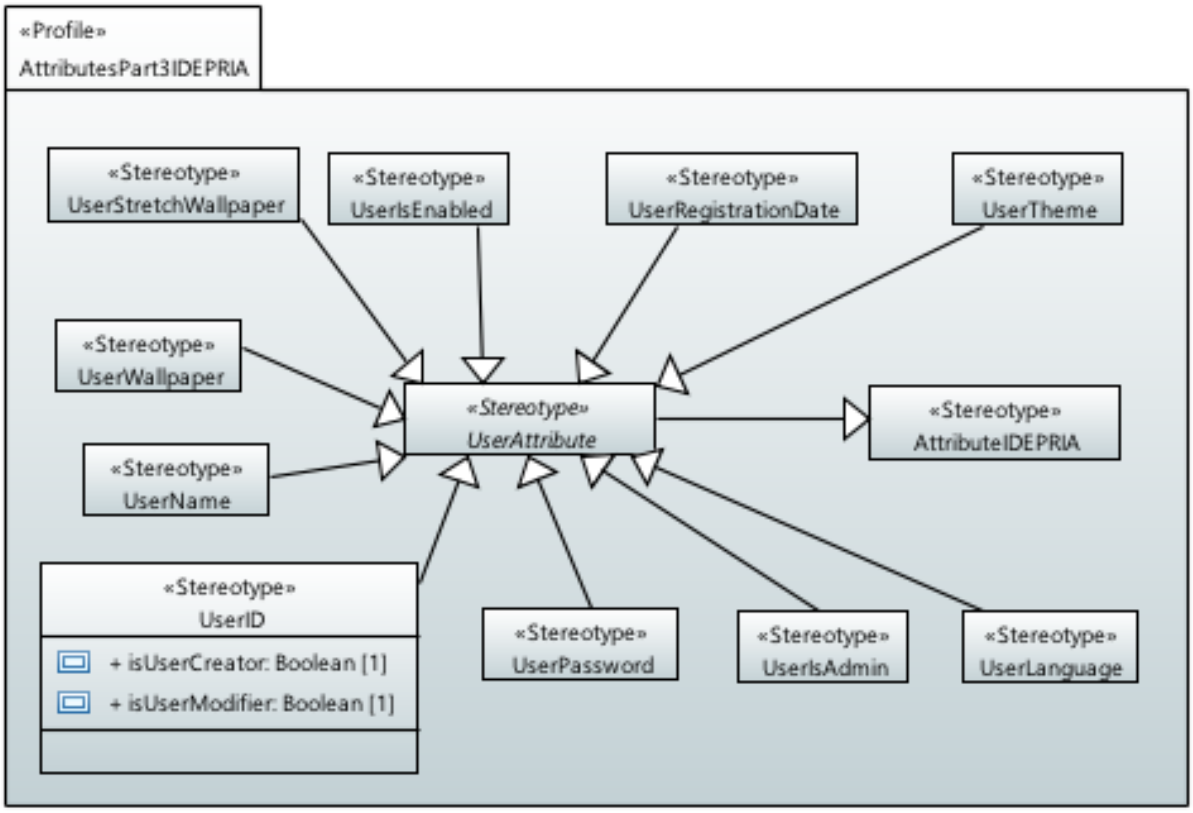

Figure 4.5: UML Profile - Attributes - Part 3

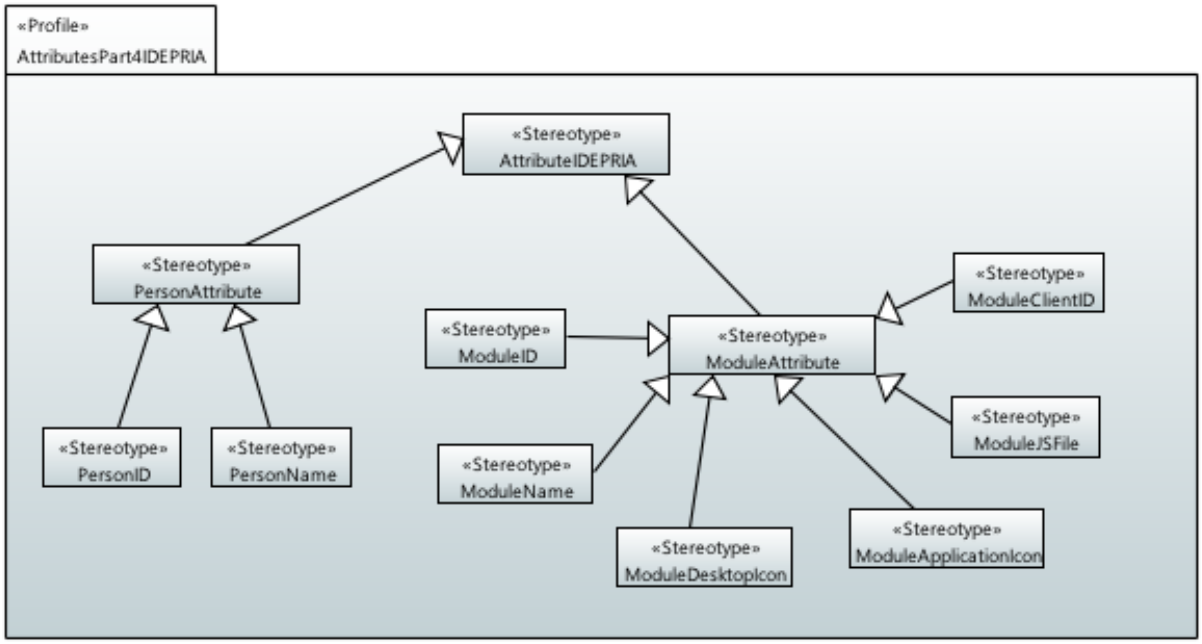

Figure 4.6: UML Profile - Attributes - Part 4

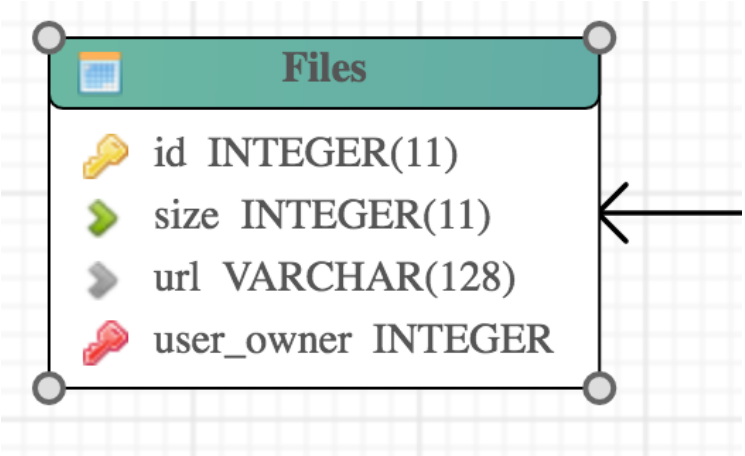

Figure 4.7: UML Profile - Custom Icons

green arrow that is to the left of the "size" attribute.

- *AttributeIDEPRIA* when *isPrimaryKey* is *true*: The primary key attributes are displayed with a gold key icon. In the example, the gold key at the left of the "id" attribute.
- *AttributeIDEPRIA* when *isForeignKey* is *true*: The foreign key attributes are displayed with a red key icon. In the example, the red key at the left of the "user_owner" attribute.
- *allowNull*: the icon is a grey arrow pointing towards the right. In the example, the grey arrow that is to the left of the "url" attribute.

Finally, Figure 4.8 shows the last extensions made in the UML Profile, the *PicketComboBox* and the *PickerWindowsSelector*, both which extend from the *Association* UML meta-class. The *PicketComboBox* stereotype will make the RIA provide an HTML combo box to pick one of many possible instances of the association-end class. The text that is going to be displayed on each entry available on the combo box is the value of the *memberIdentifier* attribute of the class (see *NormalClass* on Figure 4.2 for more information). On the other hand, the *PickerWindowsSelector* stereotype will provide a new search window that will have form fields that will permit the application of filters that will facilitate to the end-user the task of finding a specific instance of the associated class.

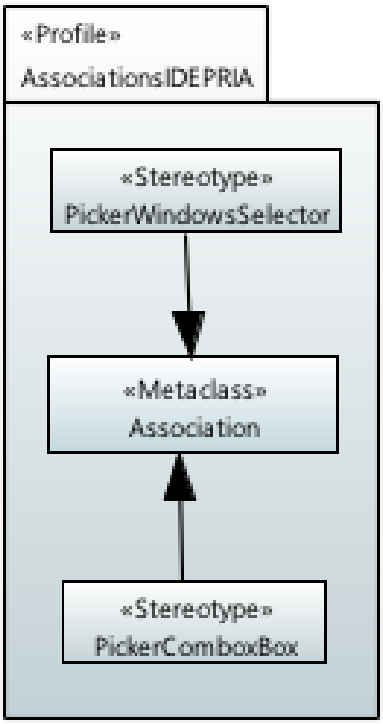

Figure 4.8: UML Profile - Associations

4.2 The IDEPRIA Tool

The developed CASE tool was named "Integrated Development Environment for Prototypes of Rich Internet Applications (IDEPRIA)" because it was built with the intention to simplify the design and development of prototypes of RIAs. IDEPRIA was designed to provide modelling support of some aspects of *UML Class Diagrams*, and more important, modelling support for the UML Profile that was described in the previous section of this chapter. However, due to time constraints for this master thesis, this tool was not designed to allow developers design all the possible UML elements described in the UML specification [O. M. G., 2004].

Also, as this master thesis intends to evaluate the usability of this Web CASE tool as a tool to design and build fast prototypes of RIAs, in this section are going to be mentioned as well the literature usability patterns followed to provide the best possible user experience to the end-users of IDEPRIA.

First, the architecture of the IDEPRIA tool follows a simple client-server pattern. The technologies and frameworks were used to build the tool were:

- **Server (back-end)**: hosts and serves the HTML/CSS/JavaScript front-end files, provides the Web services for the front-end Web application, and it is the responsible for storing persistently the data using a database engine.
  - *Python v2.7*: a multi-platform programming language that provides portability so that the back-end can run on any of the many platforms supported by this programming language.
    Website: *https://www.python.org/*
  - *Tornado v4.5*: a Python web framework that uses non-blocking network I/O, which easily allows to the back-end to scale up to hundreds of open connections.
    Website: `http://www.tornadoweb.org/`
  - *DjangoORM v1.5*: a Python middleware that allows to the back-end server to use a generic API to perform CRUD operations on many possibles database engines. Then, the source code of the back-end is not dependent on a specific database, which provides the flexibility to change the database engine in the future, without having to change the source code that handles database operations.
    Website: *https://www.djangoproject.com/*

  - *BCrypt v3.1*: a Python password hashing framework with some properties that protect from fast-hardware password cracking implementations.
    Website: *https://github.com/pyca/bcrypt/*
  - *Jinja v2.7*: a Python template engine that is used by the M2T transformation software to generate the source code of the Rich Internet Application.
    Website: *http://jinja.pocoo.org/*
  - *MySQL-python v1.2.5*: a Python framework that provides support to the DjangoORM to make API calls to the MySQL Database Engine.
    Website: `https://github.com/farcepest/MySQLdb1`
  - *PsycoPG v2.7*: a Python framework that provides support to the DjangoORM to make API calls to the PostgreSQL Database Engine.
    Website: `http://initd.org/psycopg/`

- **Client (front-end)**: the Web application that provides to developers a Web graphical user interface to design their own RIA prototypes.
  - *KineticJS v4.5*: a JavaScript framework that simplifies the development of web applications that interacts with the HTML5 Canvas. This is the core framework used by the front-end, which allows to developers to design their models.
    Website: *https://github.com/ericdrowell/KineticJS/*
  - *ExtJS v4.2*: a JavaScript framework aimed to simplify the development of data-intensive, cross-platform Rich Internet Applications. This framework was used to create the window of IDEPRIA that allow developers to customise their modelling.
    Website: *https://www.sencha.com/products/extjs/*
  - *CodeMirror v2.3*: a JavaScript framework that provides an elegant web source code editor. This was added at the end of the project as an extra feature, which allows designers to view and modify SQL triggers that can be attached to the generated source code at the generation stage.
    Website: *http://codemirror.net/mode/javascript/*

There might be another technologies and frameworks that can provide the same or even better features than the ones previously mentioned. However, this master thesis does not intend to evaluate which technology or framework is the best one, so choosing any of those that just works is enough for this master thesis. Particularly, the main reason to choose the previously mentioned frameworks is that I already had experience on those frameworks before beginning the develop-

ment of the CASE, which allowed me to save person-hours for more important aspects of this research.

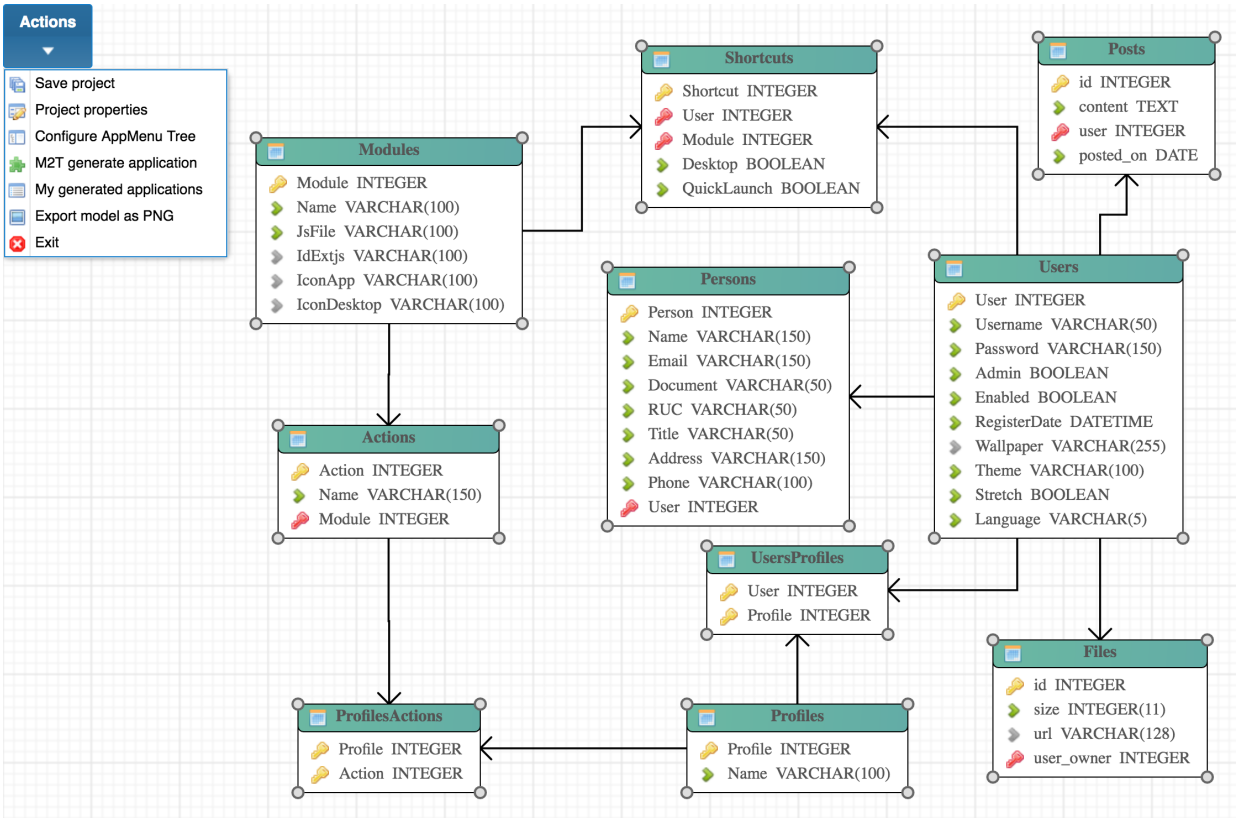

Figure 4.9: IDEPRIA Tool - Modelling example.

Figure 4.9 shows an example of a modelling of a RIA prototype with the IDEPRIA tool. At simple sight, you can see the icons of the IDEPRIA classes with its respective IDEPRIA attributes. The UML associations are represented by using the UML standard black arrow notation between classes. It is important to emphasise that the classes can be resized and relocated by dragging-and-dropping the grey circles located at the corners of each of the classes and by dragging-and-dropping the green headers of the classes, respectively. Also, the attributes can be reordered by dragging-and-dropping the attribute icon to the desired position within the class.

At the left upper corner of Figure 4.9, there is a menu named *Actions* that provides the following features:

- *Save project*: persistently saves to the project so that it can be closed and continued later.
- *Project properties*: displays a window that allows the developer to configure the name of the project, the size of the HTML5 canvas, and if the project should be periodically auto-saved.
- *Configure AppMenu Tree*: this option will allow the developer to configure a directory

structure of the classes, which will finally be the structure that the RIA will use to display the menu items for the different modelled classes.

- *M2T generate application*: displays a screen to the user that requires them to choose the desired output platform for the RIA that is going to be generated. Currently, there is only one possible output platform, but the IDEPRIA tool was designed to easily provide more output platforms in the future.
- *My generated applications*: the complete list of generated applications. Then, the developers can have access to all the different versions of the models that were transformed into the source code of the modelled RIA.
- *Export model as PNG*: creates a base 64 PNG image of the HTML5 canvas that contains the current design of the project. Sometimes is very useful to have an image of the design so it can be printed in a big and easy to analyse paper.
- *Exit*: closes the project and redirects the user to the initial project selector page.

Furthermore, there are two important interfaces provided by the IDEPRIA tool, which allow the customisation of the IDEPRIA classes and the IDEPRIA attributes, respectively. First, Figure 4.10 shows an example of the interface provided to customise a given class. At the top, it is possible to set the name of the class and to choose which IDEPRIA stereotype should be applied to the class. Additionally, it is possible to define the position of that class on the HTML5 canvas and its size (width and height) in pixels.

Figure 4.10 shows the interface provided to developers to customise the application that is going to be generated. The designer can specify the *width*, *height*, *generateCRUD*, *layoutType*, *isManyToMany*, *memberIdentifier*, *memberPrivacy*, *videoSource*, *videoType* tagged values of the *ClassIDEPRIA*, *NormalClass* and *VideoClass*, that were already described in the previous section with Figure 4.2.

From the usability point of view, this interface follows the “Error prevention” usability heuristic recommended by [Nielsen, 1993]. For example, it does not allow developers to set the *videoLink* and *videoSource* values unless the class has attached the *HTML5 Video* stereotype. Also, it only allows developers to choose an *UserID* stereotyped attribute as a possible value for the *memberPrivacy* tagged value, that is displayed with the label “Restrict instances to the authenticated UserID attribute”.

Figure 4.11 shows the interface provided to customise attributes. At the top, this window provides HTML elements that allow developers to attach tagged values to the *AttributeIDEPRIA*. At the

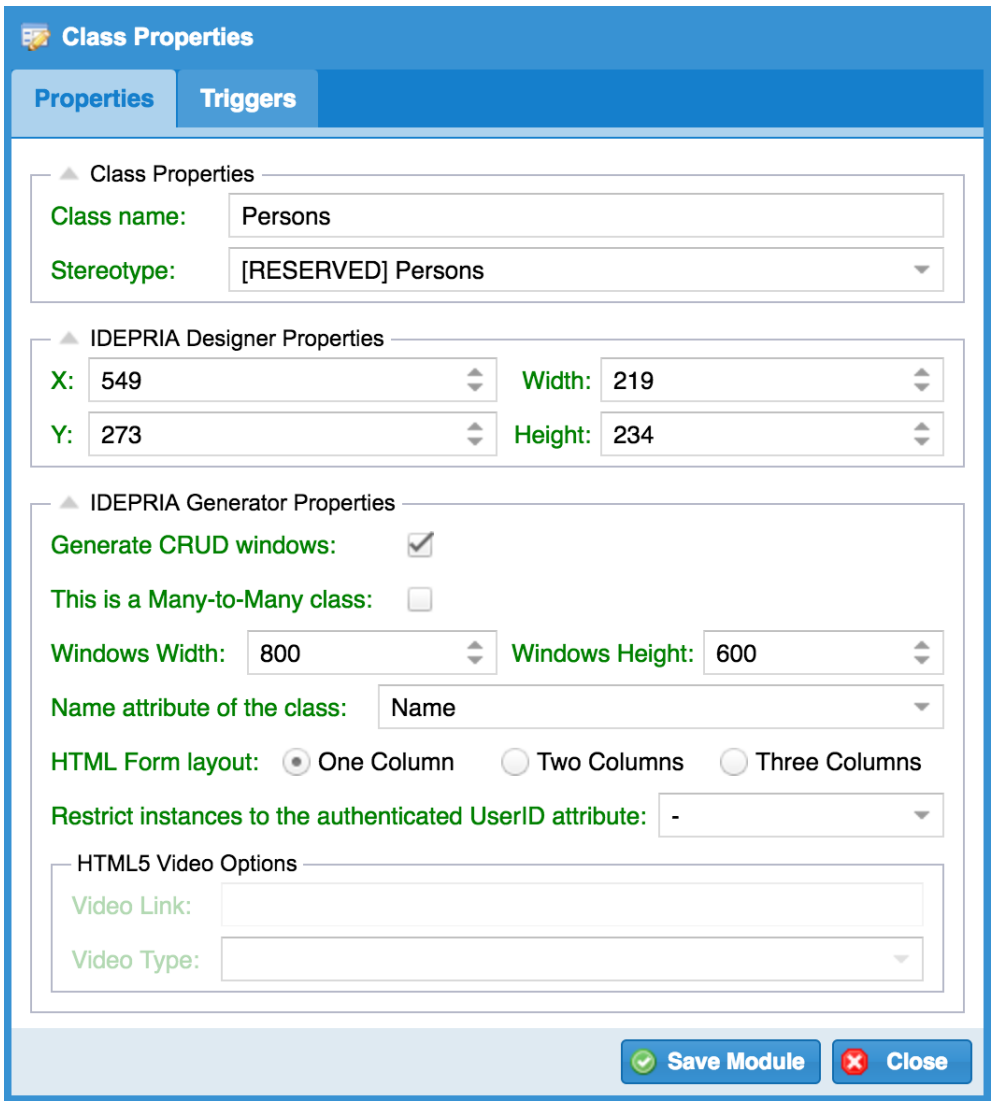

Figure 4.10: IDEPRIA Tool - Properties of an IDEPRIA Class.

bottom of the image, there is a form that allows setting the tagged values and stereotypes that are going to be used by the M2T software to customise the RIA that is going to be generated. Additionally, the example shows an attribute that is a foreign key associated with the *User* attribute of the *Users* class, and also, that the RIA will provide a *searcher interface* to permit to the end-user to find a specific *Users* instance.

Notwithstanding the fact that there are too many (twenty) form elements in Figure 4.11, the *similarity Gestalt principle*, that establishes that objects that look similar should be presented as a group, has been followed to improve the user experience [Koffka, 1935]. At the top, all the check boxes are grouped at the right, and the other combo and text boxes that look similar are grouped to the left. Similarly, at the bottom, the four combo boxes are grouped in the upper section, while all the other six check boxes are located and aligned together at the bottom.

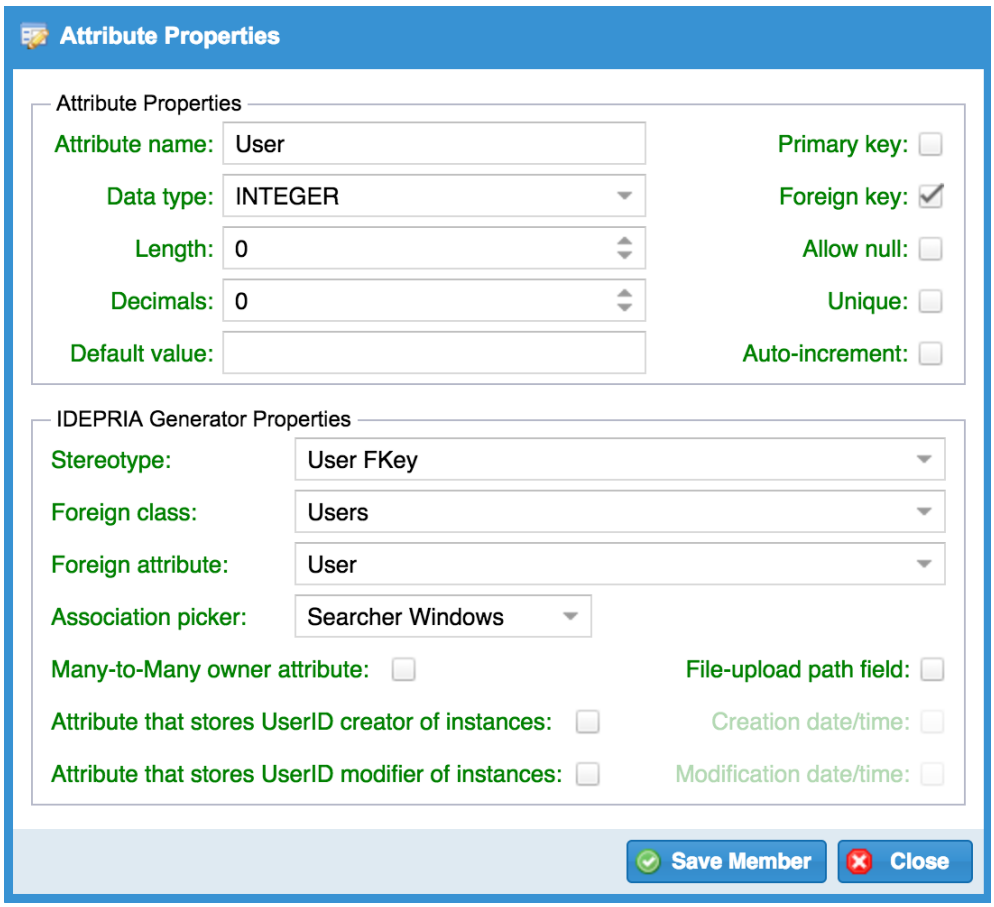

Figure 4.11: IDEPRIA Tool - Properties of an IDEPRIA Attribute.

The Nielsen's “consistency” heuristic recommendation can be clearly confirmed by a single inspection of Figures 4.10 and 4.11 [Nielsen, 1993]. Also, the icons described in the previous section were added in order comply with the “recognition rather than recall” Nielsen's heuristic. This is very useful because end-users do not have to remember which attributes are or not primary/foreign keys, or if they can have or not null values because they can recognise them at any moment by just looking at the canvas.

Finally, Figure 4.12 shows how can be obtained the ready-to-deploy source code of the prototype of the modelled RIA. The developer has to choose which output platform, and then, after pressing the “Generate” button, a download link with the zip file containing the source code will appear. Also, to make it clear to the reader how this source code generation occurs in practice, the next section of this chapter will describe the M2T software implemented exclusively to transform the models into the source code.

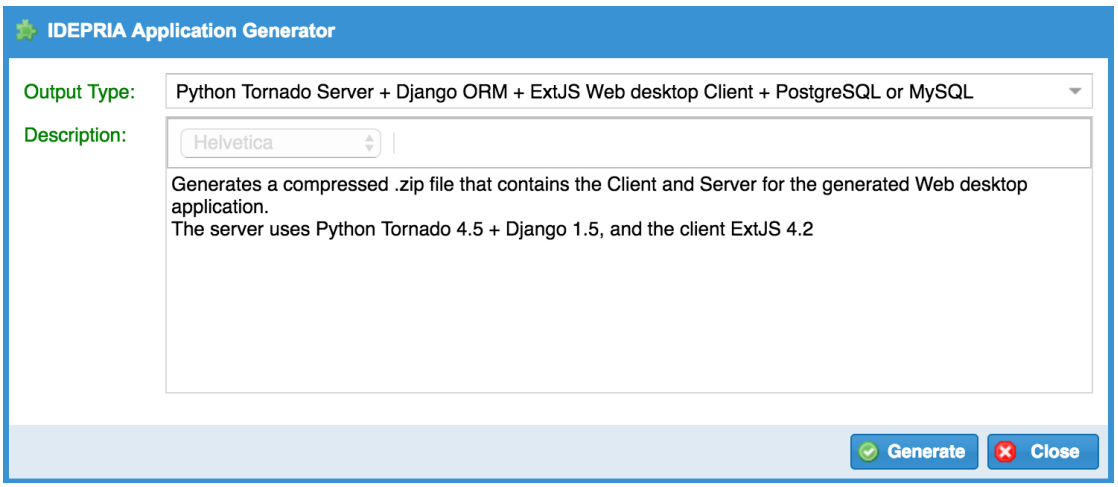

Figure 4.12: IDEPRIA Tool - Application Generator.

4.3 The M2T Transformation Software

The model-to-text transformation software is the part of the IDEPRIA tool that reads, process and then generates the source code of the RIA prototype that was designed by the developer. This software was designed by keeping in mind always the *flexibility*, which means that the software can be adapted to possible future changes in their requirements. This property is very important for the M2T software because it would be undesirable to have to modify its core source code every time we wanted to add a new output platform.

Remembering from the Chapter 2, there are mainly two ways of performing code generation, through programming languages, and also through M2T transformation languages. Given the drawbacks explained of the programming languages based approach, and the benefits provided by the template-based approaches, I decided to build a template-based M2T transformation software to make IDEPRIA able to transform the DSML models into source code.

The literature review also showed that there are too many template engines, and more specifically for the Python programming language (used to build IDEPRIA), the Python community itself provides a website[4] where they list more than fifty templates engines that work with Python. Even though that possibly many of the engines listed on that website will be enough to perform the code generation required by IDEPRIA, I decided to use the template engine *JINJA2*[5] because it was already used in the past for a scientific evaluation and it was remarked for its ease of use and flexibility [Orban, 2011].

[4] Python template engines: `https://wiki.python.org/moin/Templating`
[5] JINJA 2 template engine: `http://jinja.pocoo.org/`

Now that the chosen M2T transformation approach and the template engine have been justified, the developed M2T transformation software is going to be described to show to the reader the process that generates the source code of the prototype of the modelled RIA. Summarizing, the transformation process has five stages:

1. *Receive input parameters*: the M2T transformation software requires two parameters: the identifier of both, the desired output platform and the UML class diagram. These are mandatory parameters because otherwise, the translator would not know what models to transform, and what kind of output to generate.

2. *Model parsing and configuration parsing*: given the identifier of the diagram as input, the M2T transformation software loads it from the database and parses to an internal representation so that the data structure and its contents can be later fed to the template engine. Additionally, given the identifier of the output platform as input, the configuration file that guides the M2T transformation software is loaded from disk and parsed.

3. *Static files duplication*: first, a temporary output folder is created to put all the static files and generated source files. Then, the configuration of the chosen output platform could state that there are static files that should just be copied to a given destination inside the temporary folder. If this is the case, the static files are duplicated from the source location to the destination folder specified in the configuration.

4. *Dynamic files transformation*: the M2T transformation software iterates over each entry of the configuration file, and for each entry, the entire UML diagram is processed. At the end of the processing of each configuration entry, zero, one or many source code files could be generated, depending on which options were set in each entry.

5. *Final compression*: when the transformation ends, all the files copied/generated are compressed into a zip file, than then is moved to the IDEPRIA web server so it can be downloaded from a Web page provided by IDEPRIA.

Before describing this M2T transformation software in more detail, it is important to emphasise that the idea of using a configuration file to guide the transformation process and having static/-dynamic files/templates to generate the source code, was taken from [Marco Brambilla, 2017] and it is not a contribution of this master thesis. However, the implementation of those ideas was developed only by myself.

Appendix B shows an example the M2T configuration file. This file that guides the transformation process is the one that permits that multiple possibles output platforms could be obtained

from a single modelled *UML Class Diagram*. The developer should just use this transformation software twice, by passing the two different configuration identifiers as input parameters. A given configuration file is defined using the JSON[6] format, and this notation was chosen because it is a lightweight one that does not require a lot of tags, and because there are freely available parsers of this notation for many different programming languages. The possible keys that a given configuration file should have are:

- **id**: integer unique identifier of this configuration file. This is the field that should match the input parameter of the M2T transformation tool in order to know which static files to duplicate and which templates should be transformed.
- **name**: short description of the output that this configuration file intends to generate.
- **shortname**: short string that represents the name of the temporary folder that is going to be created in order to put the generated files.
- **description**: HTML string that describes in detail the output that will be obtained after running the M2T transformation software with this specific configuration file. This field is showed in the IDEPRIA tool as a description of what is going to be obtained as output.
- **static**: it is an array of all the static files that should be duplicated to a relative path inside the temporary generation folder. For instance, the images, CSS and JavaScript files of the ExtJS framework are static files. This *static* might contain zero or more JSON objects that must contain the following three keys:
  - **type**: that specifies if it is a file or a folder (with all its contents) what should be duplicated. Possibles values: “file”, “folder”.
  - **src**: relative path of the source file/folder that should be duplicated.
  - **dst**: destination folder to put the file/folder specified in *src*.
- **templates**: array of zero or more independent JSON objects that contains JINAJ2 templates that should be processed by the template engine. Each of the objects inside this array must have the following two keys:
  - **src**: relative path of the template that is going to be processed.
  - **dst**: destination file path of the file that is going to be generated after processing the *src* template.

[6] JSON notation: `http://www.json.org/`

- **permodule**: array of independent templates, in which each template will be processed in many rounds. This element allows the generation of source code files that depend on each of the modelled classes because, in each round, a given class of the modelled diagram will be used as input for all the templates added to this array. Each of the JSON objects of this array must have the following keys:
  - **tpl**: path to the JINAJ2 template that should be transformed by using as input each class of the diagram.
  - **dst**: destination file path of the transformed template defined in *src*. It is important to mention that the string "*modname*" can be used as value in this *dst* key to specify that this string should be replaced with the exact name of the class used to process the template. This specification allows to the generated source code files to have the name of the classes, which permits readable output filenames instead of filenames like *transformation1, transformation2, transformation 3*, etc.
  - **options**: array of named options that defines a specific behaviour while processing this specific template element. Each element contains the key "name", which could take the following values:
    - **capitalise**: it defines that the class name should be capitalised (set the first letter as capital) before replacing it with the previously defined "*modname*".
    - **excludem2m**: all the *NormalClass* stereotyped classes that have the tagged value *isManyToMany* as *true* will be skipped and will not be processed with this template.
    - **onlym2m**: all the *NormalClass* stereotyped classes that have the tagged value *isManyToMany* as *false* will be skipped and will not be processed with this template.
    - **readcrud**: process only the classes with the tagged value *generateCRUD* as true will be processed with this template.
    - **excludevideo**: all the *VideoClass* stereotyped classes will be skipped and will not be processed with this template.
    - **onlyvideo**: only the *VideoClass* stereotyped classes will be processed with this template.

The previous configuration file structure was defined in that way to minimise the amount of

information required to the developer of the file. In other words, when specifying the static files, it is only required to specify the source and destination paths and the if the source is a file or a folder. In the same way, when processing individual templates, only the source of the template and the destination file path of the file that is going to be generated after processing the template. The third possibility was added because while designing this M2T transformation software, I detected the need to process a given same template with all the different UML classes modelled in the diagram. Additionally, depending on some tagged values and stereotypes, only specific UML classes should be processed with specific templates, and this is the reason why the *options* array was added.

Nevertheless, many of the possible options available define that only some classes that have specific tagged values assigned should be processed, for instance, "onlym2m" means process only classes with *isManyToMany = true*. Then, the fact of defining new names to represent specific tag names with specific values may be confusing for the developer, and a better approach might be obtained if the *options* array is changed to just accept stereotypes or tag names with their respective values. This approach would not require developers to learn new tag names, and it will be a generic solution in case that new tag names or stereotypes are added to the UML profile in the future. Also, this change would not imply a redefinition of the entire structure and can be easily made in a future version of this software while keeping backward-compatibility by allowing the already defined keys.

Furthermore, the *JINJA2* templates are a very important part of this M2T transformation software, because they are the ones that are processed one or multiple times to generate the files that contain the source code of the modelled RIA. The JINAJ2 template engine provides a website[7] with an easy-to-understand documentation that describes the engine features and how those can be achieved. For instance, the *custom filters* feature of *JINJA2* permit calling to custom defined Python functions from inside the templates, which means that the generated output can contain any information that is feasible to compute with the Python programming language. An example of a custom filter defined is the Python function "normaliseName", which receives as input any string, and returns a string that contains only the letters and numbers of the received string. This function is very useful to transform an invalid variable name string to a valid one for the given output programming language.

To make it easy for the reader to understand how these templates are transformed into source code, Appendix C shows one of the templates used to generate a portion of the source code of the modelled RIA.

[7] JINJA 2 documentation: `http://jinja.pocoo.org/docs/latest/`

To summarise, this M2T transformation software is mainly guided by a given chosen configuration file that specifies that some static files/folder should just be copied, that there are some templates that should be processed just once while having access to the entire UML model, and that there are templates that will be processed multiples times while having access to just one UML class each time. Finally, when there are no more static files to duplicate, and there are no more templates to process, it can be said that the generation of the source has finished, and that is time to just deploy the application.

For this master thesis, there was no point in defining multiples configuration files to make able to IDEPRIA to generate multiple versions of the same Rich Internet Applications but for different platforms/frameworks. This is mainly because the IDEPRIA Web tool is the subject of evaluation rather than the prototype of the Rich Internet Application that the IDEPRIA tool will generate.

Chapter 5

Case-study evaluation

The chapter describes the case-study performed to evaluate the IDEPRIA Web CASE tool developed for this master thesis. The case-study was performed by following the framework described in [Runeson et al., 2012].

The first section of this chapter presents all the elements that were taken into account when designing the case-study, the planning performed before commencing the case-study, the kind of data that was collected, the strategy that was followed, the threats that could invalidate the case-study and the countermeasures that were applied to minimise the effects of these threats.

The second section describes the execution of the designed case-study and summarises the data collected from the *Think-Aloud* sessions and the interviews.

Finally, the third section shows the analysis performed on the collected data and the conclusions of the case-study.

5.1 Case-study design

5.1.1 Rationale for the study

The reason for undertaking this case-study is to make a novel contribution to the Model-Driven Development area. More specifically, this case-study was designed to evaluate the usability of the IDEPRIA Web CASE tool that allows developers to design and build Rich Internet Application.

5.1.2 Purpose of the study

This case-study expects to obtain results that show that the IDEPRIA MDD tool, which just requires one extended *UML Class digram* that is expected to be familiar to a person with a background in Informatics, is a well-suited and usable solution for developing prototypes of Rich Internet Applications.

5.1.3 Theoretical framework

The theoretical framework of the case-study was already described in the "Background" chapter. Please refer to that chapter for more information about the concepts mentioned in this case-study.

5.1.4 Research question and hypotheses

- *Question 1*: are Web Model-Driven Development CASE tools that require just one familiar UML class diagram a usable solution for modelling and developing prototypes of Rich Internet Applications?
- *Hypotheses 1*: IDEPRIA is a Web Model-Driven Development CASE tool that provides a well-suited and usable solution to the need for designing and building prototypes of Rich Internet Applications.

5.1.5 Cases and unit of analysis

According to [Runeson et al., 2012], the case-study is categorised as a *Holistic Single-Case study*. It is *holistic* because there is only one defined context, and it is *single* because there is only one unit of analysis [Runeson and Höst, 2008]. The *context* is the RIA prototype that was asked to the participants to design and develop, and the *unit of analysis* is the *Think-Aloud* protocol followed by the subjects while using the IDEPRIA tool.

5.1.5.1 The Think-Aloud protocol

The *Think-Aloud* protocol is a technique for examining the problem-solving skills of the participants of evaluation [Erikson and Simon, 1984]. This technique requires people to say out loud everything that they are thinking and trying to accomplish, so they thoughts can be externalised

[Erikson and Simon, 1984]. This protocol has been used already many times to evaluate the usability of software and websites (e.g.: [Bin Ahmad and Iahad, 2013], [Stefano et al., 2010]), because it shows to the evaluator the exact steps made by the subject to accomplish a task. Then, if a task is being difficult to achieve by too many participants, it can be concluded that the interface provided might need to be redesigned to see if better results can be obtained.

The three participants of the case-study that followed the *Think-Aloud* protocol were students of the Master of Science in Computer Science program of the University of Edinburgh. Each of the participants was instructed in the same way and performed the same three required tasks while following the *Think-Aloud* protocol.

[Wallace et al., 2002] proposed to record the audio of the *Think-Aloud* sessions, so it possible to analyse the record many times to find missing problems that were not annotated during the sessions. Before beginning the evaluation process, the three students agreed to be audio-recorded and video-recorded and then an informed consent form was signed by them. The consent form used is shown in Appendix D, and it is a shortly modified version of the *Think-Aloud* consent form used in this doctoral thesis [Phillips, 2014].

The first task of the *Think-Aloud* session was the most simple one and essentially asked the participant to use the IDEPRIA tool to create two classes, then to add some specific attributes to each class, and finally, to add one association to from one class to the other one. This task was aimed to help to the subject to familiarise with the tool, while at the same time I was collecting information about the level of difficulty for the subject to accomplish the task with the IDEPRIA tool.

The second task required to each of the three participants was about requiring the participants to read the requirements of a quite simple Rich Internet Application, so they can then use the IDEPRIA tool to model a possible solution to those requirements. This task is more complex than the first one, but also the most important task of this evaluation because here they will have first to think how to solve the problem and then they will have to interact in different ways with the IDEPRIA tool to design their solution properly. Additionally, relevant data can be collected from this task, such as how easy is to customise the Rich Internet Application and how usable is the interface provided to configure properties of the classes and attributes required for solving the requirements.

Finally, the third task was about asking the participants to generate the ready-to-deploy source code of the Rich Internet Application that they modelled. This task was designed so I can observe how easy is for them to generate the Rich Internet Application with the IDEPRIA tool.

At the beginning of the definition of this *Think-Aloud* protocol, there was an idea of having a fourth task that would have asked the participants to deploy and use the Rich Internet Application. However, this task was removed from this case-study because, as the only output platform currently supported by the IDEPRIA tool is a RIA that uses Python for the back-end and ExtJS for the front-end, the results of the task will be subject to much noise depending if the participants already had or not experience with those technologies. For academic purposes only, I deployed the source code that they generated with IDEPRIA, so they can see and interact with a Web browser the prototype of the Rich Internet Application that they have modelled.

The exact definition of each of the three summarised tasks can be found in Appendix E.

5.1.6 Methods of data collection

The first method of data collection was the *direct observation in controlled environment*, that was performed with the *Think-Aloud* technique [Runeson et al., 2012], which was already described in the previous sub-section.

The second method to collect data was the interview, in which the participants shared all the ideas related to the usability of the IDEPRIA tool. The interview sessions followed the *timeglass model* [Runeson et al., 2012], which means that the interviews began with open questions, then specific questions were asked in the middle, and towards the end of the interview, an open discussion was established again. More specifically, these are the two specific questions that were asked to the participants:

1. **Do you believe that the modelling elements provided by the IDEPRIA tool are enough to model just prototypes of Web Applications? Why?**. This question was made to understand the point of view of the participants about the completeness of the designed solution, which will contribute to well-suited tool research question of this case-study.
2. **How much effort and time did it take, or do you think it would take, for you to feel confident using the tool to build something useful?**. This question was made to obtain an average feeling of the participants about how easy was to use the tool with the available DSML elements.

Finally, I need to mention that a paper-based questionnaire was designed and prepared to obtain quantitative information from the participants. However, as I have only managed to get three participants for the case-study, the results would have been statistically unreliable due to the low

number of samples. Then, I did not use the questionnaire in this case-study; nonetheless, it could be beneficial for further researches with more available time and more participants willing to help.

Appendix F contains the designed *paper-based questionnaire* that I would have used in case of having more participants. The questionnaire contains close-ended and open-ended questions aimed to measure if the IDEPRIA tool was a well-suited and usable solution for designing and developing prototypes of Rich Internet Applications. As the IDEPRIA tool provides a computing support that permits modelling prototypes of RIAs using the modelling language described in the previous chapter, this questionnaire also evaluates at the same time if the modelling language designed is a well-suited and usable solution for designing and developing RIAS. [Schalles, 2013] provides an empirical framework with a questionnaire aimed at the evaluation of the usability of modelling languages. Then, the second page of the questionnaire is a slightly modified version of a questionnaire provided in [Schalles, 2013, p. 59], because it provides accurate questions aimed to understand the usability of a given modelling language.

5.1.7 Data source selection

The data was collected from participants that can be considered as good actors for the IDEPRIA tool because they took the "Software, Design and Modelling" course at the University of Edinburgh, and thus, *UML Class diagrams* are expected to be familiar to them. This selection of participants was made to avoid having experts as participants, which could probably manage to use complex systems and also to avoid having students that do not know what a *UML Class Diagram* is.

5.1.8 Legal, Ethical, and Professional Issues

This dissertation was written as a requirement to obtain a Master of Science degree at the University of Edinburgh, so all this research strictly follows the *School Ethics Code* and *ethics regulations* that are defined in the documents provided at the *Informatics Ethics* website[1] of The University of Edinburgh.

Given that this case-study involves a *Think-Aloud* evaluation and that there is just a minuscule chance of harm that could be done to the participants, after my thesis tutor approved this *Think-Aloud* evaluation, the University required me to send the *level one self-assessment (part C)* of the

[1] University ethics procedure: `http://www.ed.ac.uk/informatics/research/ethics/procedure`

informatics ethics form. Then, I received an acknowledgement that enabled me to perform this case-study as part of my thesis at The University of Edinburgh.

From the participant's points of view, before they started the evaluation, they all agree to help me with this research by signing the consent form shown in Appendix D.

5.1.9 Threats

Case-study threats can affect the validity of the results obtained, so they should be analysed beforehand to avoid obtaining invalid results [Runeson et al., 2012].

Tables 5.1 and 5.2 show the detected threats that could invalidate the case-study, and the counter-measures that were taken into account to minimise the possible effect of those threats.

Table 5.1: Threats to the case-study and countermeasures applied - Part 1

Threats	**Countermeasures**
Participants might not know how to be a participant of a Think-Aloud session.	Participants were asked if they knew the protocol and all of them knew how to do it. I reminded them all the important aspects of preparation for the evaluation, and that when they stayed in silence while using the tool, I would ask them the recommended questions [Runeson et al., 2012]: What are you trying to achieve now? What are you currently thinking?
Participants might stop the evaluation in the middle of the process.	Try to obtain evaluation agreements from at least three Informatics students as a backup strategy. Provide to participants a well-suited environment with drinks and snacks.
False interview answers from the participants.	The results were cross-controlled with the data collected from the Think-Aloud session and the interview.
Different level of skills between the three participants could provide evaluations with opposite results.	Students of the same master program at the University of Edinburgh were chosen, which were expected to have the same expertise level.
The complexity of the RIA required to design and build for the Think-Aloud sessions could be a problem bigger than the usability of the IDE-PRIA tool.	First, a simple task was asked so the participants could feel confident with the tool. The requirements of the RIA that they should design had an easy-expected difficulty for a Informatics'master student.

Table 5.2: Threats to the case-study and countermeasures applied - Part 2

Threats	Countermeasures
A very low number of participants (three in total).	It was not easy to find Informatics students that wanted to participate in the evaluation because most of them are doing their projects as well. This particular thesis was very challenging due to all the tasks that were done: development of CASE tool, DSML, M2T transformation software, Case-study evaluation. Then, the results obtained in this thesis could just provide a general idea of the hypothesis that I was trying to prove rather than a strong confirmation of the hypothesis. However, the results provide a useful starting point for a future research project.
Participants with a lack of experience with UML modelling and the Model-Driven Development approach.	Participants were asked beforehand if they had at least academic experience with UML modelling. Participants that do not know UML are not suitable to take part in this case-study because the hypothesis states that UML class diagrams should be familiar to the developers.
Preparing a biased requirements specification for the RIA that the participants should develop with the IDEPRIA tool.	This was minimised by checking with my thesis tutor the specification previous to the execution of the case-study evaluation.
The participants and I were not native English speakers so there was the possibility of miss-interpretations on what they understood as requirements of the Think-aloud tasks and on what I understood when they talked aloud.	The definitions of the tasks were checked beforehand with an informatics English native speaker. The audio and video recordings were the solutions to this, because I was able to go through them many times to understand what they were trying to do at any given moment.

5.2 Case-study execution

This section describes the prior preparation made for the *Think-Aloud* sessions and then summarises each of the three sessions. Then, this information is analysed to obtain the results of the evaluation. Also, it is important to mention that a quantitative analysis was not performed due to a small number of participants (results would have been statistically unreliable).

5.2.1 Preparation for the Think-Aloud sessions

The whole evaluation process took place at one of the University students' accommodation study room, which was booked in advance to avoid any possible interruptions. As students spend much time in the library and classrooms, the study room provided a familiar environment that made it easier for them to be comfortable. Drinks and snacks were provided before, during and after the evaluation in case they felt hungry or thirsty.

An Apple Mac OSX Sierra laptop with a Chrome Web browser version 59.0 was provided to the participants so they can accomplish the *Think-Aloud* tasks. For audio/video recording purposes, a GoPro Hero 5 Black camera (with microphone) was used to record each session. I used a paper-based notepad to takes notes on both, the difficulties that they had while using the CASE tool and the tasks that they accomplished easily without much trouble.

Before the beginning of each session, the IDEPRIA tool and the GoPro microphone/camera were tested to ensure that everything works properly during the evaluation. Then, a printed version of the tasks was given to each of the participants, so they could start working on the tasks, one after another. None of the participants met each other at any given moment during the evaluation, because they all were scheduled for different date/times.

As mentioned in Chapter 4, the defined DSML has some requirements that the modelling must comply with. To simplify this process, when starting a new project with the IDEPRIA CASE tool, the tool generates a starting model with all the required elements, so the developers do not have to worry or even know about them. A brief instruction on these was given to the participants before the beginning of the tasks. Otherwise, they would not know what were the classes that were already available at the beginning of tasks.

5.2.2 Think-Aloud sessions

All the participants are students pursuing a Master of Science in Computer Science at the University of Edinburgh, and they confirmed that they had experience on modelling *UML Class Diagrams*. They finished the *Software Design and Modelling* course at the University, so they have experience with the Model-Driven Development approach because they had generated Java source code with one of the tools that they had used in the course.

5.2.2.1 Think-Aloud session of the first participant

In the first task, the participant moved the mouse to the *Actions* menu to try to add the first class, but he could not find a way to add a class from that menu. Then, after thinking for about 20 seconds, he finally tried the right-click button on the canvas and found the menu item that allowed him to create a class. The participant easily put the name of the class in the appropriate box, and then he wondered what were the stereotypes available, and just skip that section. Then, he looked to all the other elements available to customise the class, and he had no problems with them. However, he mentioned that the confirmation screen that appeared after each successful action was annoying because he had to click on the "OK" button each time. Also, he mentioned that it was very uncomfortable the fact of having to close the interface that allowed him to create a class to start adding attributes to that class.

Later, when the task required him to create an attribute, he easily found the appropriate menu by right-clicking on the class that he had created. The interface provided was confusing to him because he did not know what was the length field, which had an initial value of zero. Then, after seeing the possible data types, he realised that the length was for the amount of characters or digits of the current attribute. Once he understood this, he added and configured all the attributes easily.

Another problem was raised when he tried to add an association because he was expecting the CASE to require him to choose the other class to associate with, and the tool required him to choose the data type and length for the associated member again. Finally, the last part of the task required him to configure the association attribute to have automatically the ID of the authenticated user, and he first opened the properties interface of that attribute, and then started looking for a valid configuration. Then, when he reached the end of the interface, he found the check box that should be ticked to set the *isUserCreator* stereotype.

In the second task, the participant started looking at each of the starter classes and attributes, and

when he saw the *Users* class he understood that some of these classes already solved the log-in requirement. However, he did not know what were for the *Modules* and *Shortcuts* classes. Also, when he was adding the attributes and classes to solve the tasks, he mentioned that it was a bit awkward having to choose, for instance, the data type *VARCHAR* for an attribute that was required to be *a short string*, according to the definition of the task. Additionally, he suggested that the *Person* class should be renamed to *UserPersonalInformation* or something different because it was confusing the fact of having two similar classes, *Users* and *Persons*.

Later, when the task asked him to configure specific behaviours, like the auto-setting of the creation date of the submissions, he was easily able to set up that kind of specifications. Then, when he created the many-to-many association between users and courses, he properly configured the *isManyToMany* stereotype, but he did not why the CASE showed him the option to put the *isManyToManyOwner* as a customization of the associated attributes. Finally, to solve the video player requirement, he remembered from the first task that he had seen a Video stereotype, so he was able to configure the class properly. He suggested that video player stereotyped classes should have a customised video to identify them more easily.

In the last task, he clicked at the *Actions* menu, then clicked on the "M2T generate application" menu item, and then he requested the generation of the RIA. At that moment, an error appeared to the participant because the classes did not have a value on the *memberIdentifier* required tagged value. The participant left the current interface and then set a valid member identifier to each of the created classes. Then, he requested the generation of RIA and downloaded the compressed zip file to finish the last task of the *Think-Aloud* session.

5.2.2.2 Think-Aloud session of the second participant

This participant started the first task by clicking on the *Actions*, and then he found no way there to add a class. Then, he continued looking at everything on the screen for 37 seconds and just then he found the appropriate menu by pressing the right-click mouse button. Then, he added easily the first class without even reading the entire screen. Later, to insert the first attribute, he did not have much trouble because he understood from the beginning what were the data types, length, decimal and default value field. However, he was confused about the fact of having to tick or not the *isUnique* stereotype for the ID attribute of the User, given the fact that the attribute was already ticked as a primary key. After thinking a little about it, he did not tick the check box. While trying to see the maximum number of characters for a field, he complained about the HTML spinner that allows to increase/decrease the numeric length field. Also, he forgot to set

the data type of the name attribute of the *User* class, and the CASE tool did not show him an error about the mis-configuration when he saved the changes. The second class was easily created by the participant.

To create the first association, he easily found the "Add association" menu item, but he was very confused because the title of the interface provided to add the association was "Attribute properties" rather than "Association properties", and also because it was the same interface that he had used when creating previous attributes. After seeing that title, he first though that there was a bug with the CASE that did not show him the appropriate interface to create the association. Then, he closed the interface and tried to add again the association, and as the same interface appeared again, he assumed that he has to just look better to that interface to achieve the task. Finally, he found at the end of the interface the two combo boxes that allowed him to define the association by choosing the associated class and attribute.

He started the second task by looking at the *User* class, and after seeing that the username and password attributes were there, he just assumed that the log-in feature was already done. Then, he created first three empty classes *Course*, *Submission*, *Membership*; then he created the associations between the *Membership* class and both, the *Course* and *User* classes. Later, he added the association between the *User* and the *Submission* class to have a way to identify which user submitted the assignment and for which course. He accomplished this task easily, and then remembered about the many to many check box that he had seen, so he opened the *Membership* properties interface and ticked the *isManyToMany* stereotype. However, he did not tick the *isManyToManyOwner* check box of any of the foreign attributes created for the associations.

Furthermore, to make the system set the current user as the uploader of the submission and the current system date/time as the one for the submission, he easily opened the properties interface of both attributes and then found and tick the appropriate check boxes. For the video player requirement, he did not know how to model it. Then, he first looked at the *Actions* menu, and as he did not found a solution there, then he created a class just because there were no other options. Subsequently, after clicking on the stereotypes combo box, he realised that the video player was just a stereotype that should be attached to a given class. To finish, he chose the *mp4* video type and copied the source URL of the video.

Finally, the last task was easily accomplished by him because he had already seen the "M2T generate application" menu item in the *Actions* menu. In that menu, he chose to generate the RIA built with the Python programming language, then he read the description and clicked on the button to generate the RIA. The missing member identifier error appeared (as it appeared to

the first participant), so he closed that interface and then changed the member identifier values of all the classes that he had created. Finally, he was able to generate and download the compressed RIA source code.

5.2.2.3 Think-Aloud session of the third participant

The third participant was a woman, which started by saying that the only possible available action is to open the *Actions* menu, but there she did not find the option to create a new class. She kept looking at the interface, and after some time she gave up, and I had to intervene to explain to her that there were options that would appear after pressing the right-click mouse button. She was all right with that but shared the idea that in Web applications the right-click is not a very common because she normally uses the right-click mouse button to see available options provided by the Web browser rather than to see options provided by the website.

Later, she was able to start the creation of the first class, but she was confused about the stereotypes that were available to select, so she just left the default value to avoid making a mistake. She was also confused about the data types because the definition of task asked her a "variable length characters" and there was not a "string" type available. She guessed and picked the "VARCHAR" data type (which is the appropriate one), and then, she easily understood that in the length field she had to put the maximum number of characters that the attribute could have. Later, she complained about the spinner because it was very complicated to click the upper arrow due to its small size. Later, while creating the attribute for the identifier of the class, she was very happy about the fact that the interface did not allow her to check the primary key check box when the chosen data type for the attribute was not valid for a primary key (e.g.: a TEXT data type can not be a primary key because the underlying database engines will not permit to index that kind of large text fields). Then, before saving the new attribute, she was confused about the fact that the button to save the attribute had the text "Save member" instead of "Save attribute".

Later, she was wondering if she had to do anything special to customise the *url* attribute as a file upload HTML form element. So, after looking at the properties interface of that attribute, she decided to tick the appropriate "File-upload path field" check box. Then, she got to the part in which an association had to be created. She found the option very easily by right-clicking on the class, but then she was confused because the title of the provided interface was "Attribute properties", which was not the "Association". However, even with that bug, she just continued and tried to set the appropriate values of the association by choosing the correct values at the foreign class and foreign attribute. After that, she easily understood which check box to check

make the system store the currently authenticated user as the creator of the "Files" instance.

For the second task, she looked at all the starter classes provided by IDEPRIA as an initial RIA project, and she easily understood what was for the elements: *Users*, *Persons*, *Profiles*, *Actions*, and its associations. However, she did not understand the purpose of the *Modules* and *Shortcuts* classes. For the first part, after she saw the username and password attributes in the *Users* class, she just assumed that the authentication requirement was already solved. After that, she started solving the modelling problem by creating the appropriate *Courses* and *Submissions* classes. Then, she easily creation the association to the already existing *Users* class. She did not add a class to represent the membership of a user of a given course, and mentioned that by evaluating to which courses a given user had sent submissions, it could be possible to know to which courses a given user belongs to.

To assign the stereotypes *isCreationDate* and *isUserCreator*, she knew that she had to go to the properties interface of the "submission_date" and "uploader" attributes, respectively. In each interface, she detailed went through all the possible customisation elements, and she easily found which check box to tick for each attribute. For the file upload path, she already has done it before in the first task, so she set the appropriate *isFileUploadPath* stereotype very fast.

Furthermore, to create the video player, she already knew that she had to add the *HTML5 Video* stereotype to a given class, but she did not know to which one. Then, she just created a new class for the video player and set the appropriate stereotype, video type and source URL. After saving the class, she looked at the class and thought that something was wrong because the class just showed an empty box with nothing inside, so she expressed that a better icon for Video stereotyped classes would be great.

Finally, for the last task of generating the RIA, she had the same "missing attribute identifier" problem that the previous two participants had. Then, she returned to the modelling screen and fixed all the attribute identifier of all the classes that she had created, and then she was able to generate and download the RIA.

5.2.3 Interview sessions

To remember to the reader, these were the two specific interview questions: 1. Do you believe that the modelling elements provided by the IDEPRIA tool are enough to model just prototypes of Web Applications? Why?; 2. How much effort and time did it take, or do you think it would take, for you to feel confident using the tool to build something useful?

5.2.3.1 Interview to the first participant

The first participant started by saying that the associations in UML are between two classes, and that is was weird having to create an attribute when he was trying to create an association. However, he said that the association that IDEPRIA allows you to create is a directed association because it has an arrow icon. Then, he mentioned that for directed associations, it makes more sense the attribute creation because it should be possible to specify the field that should be referenced to the associated class. Nonetheless, he emphasised that IDEPRIA should not ask the data types or anything else than just the attribute of the associated class.

For the first question, he said that rather than asking if there were enough design elements, I should ask if there are not too many due to all the reserved modelling elements, e.g.: Users, Profiles, Actions, among others. However, even though that these all elements are required by the modelling, as the IDEPRIA project already provides all of them as an initial project, he emphasised that he did not have to create any of them. He mentioned that he knows that some very well-known frameworks for developing Web applications, such as Django, also provides some of these features in new projects. Then, he said that these starter models could be advantageous just to have them even if we do not use them at first because later we might need them.

Later, he added to the first answer that the file upload path field and the possibility to make instances dependent on the authenticated user that created them are great features that are not only useful for Web applications but to any type of applications as well. Then, after he saw the generated RIA, he was amazed by the quality and the beautiful aspect of the RIA, because he emphasised that he did not spend much time customising the RIA. Then, he concluded that for prototypes of Web applications, this is great because in a prototype developers just want to check if the requirements were meet with the created models, without having to spend much time in modelling how the final application should look like.

For the second question, he just stated that as he finished the three tasks in less than 77 minutes, he concluded that the tool is following the right path and that after fixing the issues detected (usability and missing helping information), it could become a great tool in the future.

5.2.3.2 Interview to the second participant

This participant began mentioning that the tool looked good because there were no so many options available (just an Actions menu), and thus, it is hard to become lost with the tool. Also, he mentioned that he had used Eclipse Papyrus and that it was very annoying because there were

so many menus with so many possible options and explorers (project explorer, model explorer, overviews) at any moment, which made him a really complicated task know where to begin. However, he mentioned that the right-click menu provided by IDEPRIA is hidden and that the right-click menus in Web environments are common for Web browsers'actions rather than applications'actions.

The participant answered to the first question by saying that, for modelling prototypes RIAs, there were a good average number of modelling elements. Nonetheless, he mentioned that there are many elements missing that could be needed for the modelling of full RIAs, including Web breadcrumbs, modelling of how to navigate between the different interfaces of the classes, and how to display complex data structures such as graphs/trees.

For the second question, this participant was very happy because he was able to finish all the *Think-Aloud* tasks in approximately 60 minutes. This was possible because, after just finishing reading the three tasks, he thought that the second task was more challenging than the first one. Also, he emphasised that when he used Eclipse Papyrus in the past, that tool took him about three days to have some code generation working, and with IDEPRIA he generated the application with just three clicks in less than two minutes. Nonetheless, he still believes that some tooltips or helping information would be great in IDEPRIA because it is kind of difficult to know what are for all the reserved stereotypes.

Finally, in the last open part of the interview I mentioned him that this tool only required him just one diagram to be able to generate a RIA prototype, and when I was talking, he interrupted me to mention that the minimalist approach of IDEPRIA (not so many options available and requiring just one single diagram) is one of its best characteristics because that makes possible to easily have something working without being lost in the process.

5.2.3.3 Interview to the third participant

The last participant started by mentioning that it is urgent for the IDEPRIA tool to have, at least in the *Actions* menu, the options to add classes/attributes/associations because the menu that appears after pressing the right-click mouse button is hidden and there was no way for her to know how to add the first class.

Then, she answered the first question by saying that is very difficult to know if all the design elements provided are or not enough for developing prototypes of RIAs because there might be different needs for different requirements of RIAs. However, she said that having so many design

elements for all the possible requirements would decrease the usability of the tool because it would be very difficult for developers to choose the right design element for a given case. Finally, she concluded that the elements that she was able to set and customise during the modelling were appropriate and that what she saw in the generated RIA exceeded her expectations.

In the second questions, she answered that with the IDEPRIA tool she had way better results than with her previous experience with the Eclipse Papyrus modelling tool. Moreover, she said that she was able to finish the tasks of the *Think-Aloud* session in about 90 minutes, but a task of similar difficulty took her almost two days to finish it with Eclipse Papyrus. Also, she emphasised that IDEPRIA have usability issues, but the tool can still be used easily because there are no many options available all the time (in contrast with Eclipse Papyrus).

Furthermore, she said that the "missing identifier" error that appeared while she was trying to generate the RIA was annoying because the "Generate RIA" interface allowed her to select the output platform for the RIA, and just then the error appeared. In other words, she stated that the interface should not allow her to do that given the fact that she will have to close in all cases that interface to go and fix the error.

5.3 Case-study analysis

5.3.1 Qualitative data analysis

First, I checked that the data collected from the *Think-Aloud* sessions and the interviews are not contradictory, to avoid using data that could invalidate the results. Then, after reading the summaries of both, *Think-Aloud* sessions and interview sessions, I can confirm that the data collected from both sources are not contradictory.

Then, after viewing again the videos recorded during the *Think-Aloud* sessions and reading the notes taken at the interview sessions, it is evident that the participants experience many usability issues while they were using IDEPRIA to solve the requested tasks. However, not all the usability issues detected have the same severity level, and that is the reason why I decided to create the following three categories to group the issues:

1. **Low severity**: usability issues that can be easily fixed, and probably with an obvious solution. These issues still permit to developers continue the modelling without much trouble.
2. **Medium severity**: usability issues that do not have an obvious solution because they might

be solved in different ways. Nonetheless, the developers understand the issue and can continue working with the tool.

3. **High severity**: usability issues that do not permit to developers to use a given interface or design element because they do not know what is that for. Also, to this category belongs all the usability issues that do not allow to developers to continue using the tool for solving a given problem.

Given the previous categories, the next listing describes all the usability issues detected during the evaluation, the severity levels of each of them, and possible alternative solutions that could be applied to solve those issues.

Low severity usability issues:

1.
   - *Issue description*: successful confirmation dialogue message that appears after saving-/editing a class/attribute/association is frustrating because it requires to developers to click on the "OK" button each single time.
   - *Alternative solution*: remove the dialogue message when there are no errors, then, the users will just assume that the element has been successfully saved. However, if an error is detected, a dialogue message with the error should be displayed so the developer can make the appropriate change.

2.
   - *Issue description*: No possibility to add attributes to classes from the interface provided to create/edit classes. Then, the developers have to close the current class interface, go to the modelling canvas, right-click on the class, and then click again on the menu item to add an attribute/association.
   - *Alternative solution*: Instead of just providing a "save button" that just saves the element and then closes the interface to customise that element, it could be a good idea to have a shortcut button as well that changes the interface to the one that permits the creation of attributes.

3.
   - *Issue description*: the "Users"and "Person" stereotyped classes are sometimes confusing because developers do not know which one to use to represent a given user of the system.
   - *Alternative solution*: rename of the stereotyped class "Person" to "UserPersonalInformation", because that is a more clear name for its purposes. Then, the only obvious choice for users of the system would be the "Users" stereotyped class.

4. • *Issue description*: the icon provided to Video players are the same than the ones provided for classes, and that does not help developers to recognise by just looking at them.

   • *Alternative solution*: creating a new icon similar to a video player, and replace the standard class icon with this new one for classes that were stereotyped with the *Video-Class* stereotype.

5. • *Issue description*: error of invalid model appears after selecting the desired output to generate the prototype of the Rich Internet Application. Then, there is no point in letting developers choose a given output if an error will eventually appear later.

   • *Alternative solution*: this should be a process with two stages, in which the first one should be automatic by showing the errors (if any) to the developer. Then, if no errors were detected, it should be able to proceed to the second stage, in which the developer can choose the appropriate output platform.

6. • *Issue description*: attributes can have the four possible boolean combinations of the primary key and unique boolean fields. This is wrong because by definition a primary key is unique.

   • *Alternative solution*: if a developer mark a given field as primary key, it should automatically mark the attribute as unique without even giving the possibility to developers to change that fact.

7. • *Issue description*: the HTML spinner provided to change the length of an attribute are difficult to use between their arrows are very little and close to each other, which makes developers miss-clicking the appropriate arrow or the arrow at all.

   • *Alternative solution*: change to a simple text input box that allows only numeric characters between a given range.

8. • *Issue description*: the interface provided to create an association has as a title “Attribute properties”.

   • *Alternative solution*: the title should be renamed to “Association properties”.

9. • *Issue description*: the button to save the properties of a given attribute contained the text “Save member”, rather than “Save attribute” or just “Save”.

   • *Alternative solution*: Member is a term that is sometimes used in Java, but not for modelling with UML. Then, that text should be changed to just “Save”.

Medium severity usability issues:

1. • *Issue description*: confusing default values on the fields provided to customise a given class/attribute/association. For instance, the length field has a default value of zero always.

   • *Alternative solution*: never provide a default value of 0 for any field and set the different default values depending on the data type selected. For example, 250 for strings, 11 for integers, 25 for long integers, and so on.

2. • *Issue description*: the interface to create the association and the one to create an attribute are the same. Then, it is not necessary to ask many fields (length, decimals, data type) for associations because those values can be automatically obtained from the associated foreign class and attribute

   • *Alternative solution*: a new interface to create the association should be provided, without asking developers for fields that can be automatically obtained.

3. • *Issue description*: too many possible data type values for attributes and difficulty to understand the names of these data types.

   • *Alternative solution*: all the currently provided data types should be analysed to see if all of them are required. Instead of providing so many options for strings (CHAR, VARCHAR, TEXT, LONG TEXT), it could just be better to have (STRING FIXED LENGTH, STRING UNLIMITED LENGTH), which might end up being more understandable to developers because of the presence of the word "String" in the name. Additionally, the length and decimal fields should be disabled for the unlimited one, and the length field should be enabled and required for the fixed length version.

4. • *Issue description*: when the developers make a wrong configuration of a design element or when they forgot to specify a required field (e.g.: data type of the attribute), there is no any information shown about the error.

   • *Alternative solution*: do not allow to save the design element when an error is detected. Instead, show the error to the developer at that moment, and ask him to make the appropriate change (e.g.: "please set a valid data type for the attribute").

5. • *Issue description*: difficult to know at first sight how to model a video player. The only current way is to specify the stereotype "HTML5 Video" while creating/modifying a given class.

- *Alternative solution*: Create one more option in the *Actions* and in the right-click options menu that permits the creation of video players by asking just the video type and the video source, without asking many of the other common elements available for traditional classes.

High severity usability issues:

1. - *Issue description*: the list of options that appear after pressing the right-click mouse button is not easy not find. Moreover, for web applications, the options that appear after pressing the right-click are normally related to Browsers'actions, rather than Web applications'actions.
   - *Alternative solution*: add menu items to the already available *Actions* menu. These menu items should permit the creation of the core design elements: classes, attributes, associations. Also, for access to the properties interfaces of the classes and attributes, a small "edit" icon can be added to these elements to avoid having to perform necessarily a right click on them.
2. - *Issue description*: some fields of the interfaces that allow to customising the classes/attributes/associations could not be understood by the developers. For instance: the length field of an attribute, or the *isManyToManyOwner* stereotypes.
   - *Alternative solution*: provide a tooltip icon next to each of the labels of the fields. Then, if the developer put the mouse over those icons, a help information screen will show up to the developer to explain what it is for and a simple usage example.
3. - *Issue description*: some predefined stereotyped classes (e.g.: shortcuts and modules) can not be easily understood by the developers when they see them.
   - *Alternative solution*: provide a tooltip with an "information icon" that would display information about those stereotypes when the developers put the mouse over those icons.

The reader can see that the evaluation performed identified seventeen usability issues. Nine of them belongs to the low severity group, five of them to the medium severity group, and only three of them to the high severity group. These are significant issues that make it difficult to use the IDEPRIA tool. The third participant could not even continue at the beginning of the task due to hidden right-lick menu option. Nonetheless, even with all these usability issues, the participants expressed in the interviews that they were able to successfully finish the tasks in a reasonable time.

Some of the participants even compared their modelling experience with other modelling tools (e.g.: Eclipse Papyrus) that they had used and then concluded that the simplicity of IDEPRIA is one of its best attributes. Also, they said that this simplicity is due to the appropriate average number of modelling elements provided by the tool, and the fact that there are not so many available options at every single moment.

Some of the participants emphasised that starter classes and features provided by an initial IDEPRIA modelling project could save person-hours because these are features that are typically required while developing Web applications, for instance, the users class with the login and register features. Additionally, they said that it was pretty amazing the RIA that they obtained from the models that they had created.

To summarise the results obtained, the tool has at least seventeen usability issues, and there can be more than have not been detected. The three participants were able to finish the tasks that were assigned to them in a reasonable time, which means that they were able to learn and use the modelling elements to build a prototype of a Rich Internet Application. Also, three participants are not enough to prove the hypothesis, but I can conclude that the IDEPRIA tool, which has been developed from the scratch, obtained promising results because it was successfully used by three of its target users to develop a prototype of a Rich Internet Application without much trouble. Then, it might worth trying applying the solutions to the detected issues, and continue performing further evaluations to see if better results could be obtained in the future.

Chapter 6

Conclusions

This final chapter shows to the reader remarks of what I have achieved in this master thesis and some suggestions of what could still be done to continue this research.

The first section presents a summary of this research and the work done during the development of this project.

The second section describes the general conclusions obtained mainly from the feedback and analysis of the case-study evaluation.

The third section lists the main contributions of this master thesis.

Finally, the fourth section provides suggestions for further work that could contribute to the state of the art of the research area of this master thesis.

6.1 Summary of the work done

First, this research started with a review of the literature related to Rich Internet Applications and Model-Driven Development. After reviewing them, I was able to understand the importance of the prototypes of Rich Internet Applications, and how the development of this kind of application could be improved and accelerated by following different possible Model-Driven Development approaches.

I reviewed the state of the art of the current MDD approaches and its CASE tools. The information was not only obtained from formal sources of the literature, but also from each of the websites of the tools, which explained how the CASE tools worked and for which purposes they were

appropriate for. Then, I realised a simple comparative analysis between these tools, whose result was that none of the tools met all the criteria that were previously defined. After this results, I concluded that a new MDD Web CASE tool for developing prototypes of RIAs could still be a good way to contribute to the state-of-the-art.

Later, the more time-consuming part of this project started: the design and development of the Web CASE tool and its ability to generate a ready-to-deploy prototype of a RIA. Summarising, the project is the combination of these four software components:

1. **The Domain-Specific Modelling Language**: the modelling language created to permit the modelling of prototypes of RIAs. This new modelling language was defined using UML Profiles, which are just an extension of the standard UML. The Profile UML defined includes more than 100 design elements created with Eclipse Papyrus, including stereotypes, enumerations, literals and tagged values.

2. **The Web CASE tool**: the software tool named *IDEPRIA*, which was developed to provide support for the design elements defined in the Domain-Specific Modelling Language. The development of this tool included the back-end developed in *Python* with the *Tornado* framework, and also the front-end, which used mainly the *KineticJS* framework to manipulate the *HTML5 Canvas*. The final version of CASE tool consists of 71 files that contain a little more than 10.000 lines of code that I wrote in JavaScript, Python, HTML, CSS, SQL. This count number does not include any framework or external package.

3. **The Model-to-Text transformation software**: the software that transforms a given modelling project into a given output source code. This was integrated into the CASE tool, so the Web interface could provide a way to the end-users to generate the modelled RIA. This software consists of 4 files that contain around 250 lines of code in Python. This count number does not include any framework or external package.

4. **Templates for a ready-to-deploy Rich Internet Application**: as the CASE tool provided had to be able to generate a RIA, I had to first develop a sample and working application to transform it into generic templates that could then be used by the M2T software to generate the modelled RIA. This sample application included the development of a Python back-end and a front-end that used the ExtJS framework. The final template obtained to generate the RIAs consists in 56 files that contain a little more than 5.100 lines of code that I wrote in JavaScript, Python, HTML, CSS. This count number does not include any framework or external package, but it includes some lines of code taken from the ExtJS website for the front-end of the RIA.

The number of design elements of the DSML and total lines of numbers of the other three components was provided to show to the reader an approximation of the amount of work performed. Also, the CASE tool, the M2T transformation software, the templates for the RIA, and all the libraries and frameworks together consist of 334 files that contain 490.494 lines of code. All the previously mentioned source lines of code counts were calculated using "CLOCK"[1].

Once finished the development of the four core components, I started the design of the case-study, which was executed to evaluate the work done and to see if the hypothesis could be verified. The case-study was developed by following the case-study framework described in [Runeson et al., 2012]. The case-study was mainly about having Informatics students using the tool to accomplish some tasks while following a *Think-Aloud* protocol. At the end of each of the *Think-Aloud* sessions, the participants had a *timeglass* style interview with me.

All the results obtained from the *Think-Aloud* sessions and interviews were summarised and analysed to validate the status of the hypothesis defined at the beginning of this master thesis.

6.2 Main contributions of this project

I performed a global analysis of this master thesis, and I concluded that these were the main contributions:

1. A summary of the background of the topics related to Rich Internet Applications and Model-Driven development.
2. A simple comparative analysis of current MDD approaches and tools that permit the design and development of Rich Internet Applications.
3. A Web CASE tool that allows the modelling and development of Rich Internet Applications by requiring just one extended UML Class Diagram.
4. A Model-to-Text transformation software that permits the transformation of models that were created with the defined Domain-Specific Modelling Language, into a set of source code files and static libraries that together represent a prototype of the modelled Rich Internet Application.
5. A case-study evaluation which showed that even when the IDEPRIA Web CASE tool does not provide the best possible user experience, average Informatics students were able to use

[1] CLOC - counter of lines of code: `http://cloc.sourceforge.net/`

the tool for modelling and building a simple Rich Internet Application in a reasonable time. This fact does not prove the hypothesis, but it is a promising result that can be used as a baseline for further related researches.

6.3 General conclusions

RIAs are a great solution for developing Web software systems because they provide features that improve the overall end-user experience. The development of this kind of Web applications can be accelerated by following Model-Driven Development approaches, which often provide CASE tools that can transform models into the source code of these Web applications. During this master thesis, I was able to see the advantages and drawbacks of many currently available MDD approaches aimed to simplify the development of RIAs. All of these are different in many ways, which makes them an appropriate solution for some specific problems. However, after performing a comparative evaluation, I detected that some evaluation criteria were not met by all of them, including: numbers of models required, availability of a Web CASE tool, possibility to generate many possible outputs from one model, and others that were mentioned in Chapter 3.

During the development of the case-study evaluation, I discovered that the first version of my Web CASE tool had 17 usability issues. Most of these issues did not cause a really bad impact, and their solutions are straightforward, e.g.: removing the dialogue with the confirmation message that appears after saving a class/attribute. However, there were a few issues that could have a bigger impact, for instance, at first they did not know what were the “reserved” stereotypes, and as there was no helpful information that explained what they were for, the participants just avoided those stereotypes.

Notwithstanding all the usability issues that I detected in IDEPRIA, all the three participants were able to successfully finish the modelling tasks that they were asked for in the *Think-Aloud* sessions. Additionally, they were able to finish the three tasks in reasonable times. In other words, the IDEPRIA tool, even with all its detected problems, provided at least the minimum features that permitted to the participants learn and use the tool to solve a simple modelling case in a reasonable amount of time. Also, the three participants were able to see and use the RIA prototype that was generated from their models.

Furthermore, as a result of the interviews, I was able to confirm that the participants enjoyed the fact they just had to create one modelling project that looked pretty similar to a *UML Class Diagram*. They mentioned that it was the first time that they were able to create a UML model

within a Web environment, and they, as potential future real-end users of the tool, believe that the minimalist style of IDEPRIA is one of their best qualities, because they said that they had tried other CASE tools in which they do not even how to start because of the many available options at any given moment.

I would like to emphasise that, as IDEPRIA is a new tool developed from the scratch, it is the first time that it has been used as a part of a case-study evaluation. Also, as stated in the previous chapter, promising results were obtained, and I conclude that it is worth trying solving the minor issues and proposing new alternatives to the more important usability issues that were detected to obtain even better results the next time.

6.4 Suggestions for further work

Given the literature reviewed, the implementations of the CASE tool and the results obtained during this master thesis journey, this section describes possibles further works that could be done in to contribute to this research area.

First, all the usability problems that were detected during the *Think-Aloud* sessions should be analysed in depth to propose the appropriate changes that could improve the IDEPRIA tool. The case-study evaluation was performed only with three participants due to the amount of work that took the development of the four core elements of this project and the time constraints of this master thesis, then, it is vital to perform more sessions to obtain results that could lead to a more robust conclusion.

Given the fact that there are too many possible elements that could be provided by a DSML aimed for the modelling of RIAs, it could be a great idea to prepare an online Web-based survey to ask to software developers, that have experience with modelling and Web development, to rank which Web elements they believe that should be possible to model and customise with a given MDD CASE tool. The results of these survey can be used to refactor the developed Domain-Specific Modelling Language.

Mobile applications are as well another kind of applications that could be developed by following a MDD approach. Given the fact that the IDEPRIA tool is already able to generate the back-end API of the RIA, it could be a great further work to create templates that could transform the models into into source code of mobile applications (iOS, Android, WindowsPhone, among others) that interacts with the already provided back-end API. Also, the DSML could be extended

(or even a new DSML could be created from scratch) to provide design elements aimed at the modelling and development of mobile applications.

Additionally, there is another approach that might be combined with what I have done for this master thesis: *Model-based Test-Driven Development (MbTDD)*. Model-based testing is a technique that permits the automatic generation of highly qualitative tests for a given system, which could reduce the number of bugs, the testing duration, and the testing costs [Mou and Ratiu, 2012]. It could be interesting to evaluate the possibility of adding test modelling elements to the DSML to provide support for the automatic generation of ready-to-deploy testing components of the developed RIA. [Baerisch, 2010] describes how *Domain-Specific Modelling Testing* could be implemented in practice.

Finally, the fact that the IDEPRIA tool only provides support to one user at a time to edit a given modelling project categorises it as a single-user CASE. Single-user CASE tools have become a productivity bottleneck because it does not provide interfaces for simultaneous editing of models by a collaborative team [Red et al., 2014]. Then, it could be an exciting project to evaluate the possibility of having multiples users modifying a given modelling project at the same time. Then, if the advantages are worth trying, the IDEPRIA CASE tool could be redesigned and refactored to provide a real-time collaborative modelling feature.

Appendix A

Detailed RIA DSML description

This appendix describes in detail and justifies all the modelling elements included in the Domain-Specific Modelling Language shown in Chapter 4. This information was put in this appendix so the reader can refer to this information just when needed.

A.1 IDEPRIA Project

This section describes the stereotype *Project*, which extends from the UML meta-class *Project*. This *Project* stereotype, shown in Figure 4.1, represents a project in which a RIA can be modelled using the IDEPRIA tool, and has attached the following tagged values:

1. *projectID*: integer identifier of a given project.
2. *creationDate*: the creation date of the project.
3. *autoSave*: boolean that determines if a given project should be saved automatically.
4. *autoSaveSeconds*: the number of seconds that should be waited before automatically saving the project again. This value is only used if the *autoSave* attribute is *true*.
5. *width*: the width of the HTML5 canvas in which the end-user design the project.
6. *height*: the height of the HTML5 canvas in which the end-user design the project.

A.2 IDEPRIA Classes

This section describes the stereotypes that extend from the UML meta-class *Class*, which are shown in Figure 4.2. The abstract stereotype *ClassIDEPRIA*, from which extends all the other stereotypes, represents a simple extension of a UML class, but which has attached the following tagged values:

1. *classID*: integer identifier of a given *ClassIDEPRIA*.
2. *windowsWidth*: the width of the section of the RIA that will be generated, that will permit to end-users operate with instances of this stereotype.
3. *windowsHeight*: the height of the section of the RIA that will be generated, that will permit to end-users operate with instances of this stereotype.

There are two specialisation stereotypes that extend from the stereotype *ClassIDEPRIA*:

1. *VideoClass*: this stereotype is defined to simplify the modelling of Web video players, so the M2T software can generate windows that allow end-users to play/stop a given video inside the RIA. This stereotype requires two tagged values to work: the *videoSource*, which should contain the URL to the video so it is possible to know which video should be played; and the *videoType*, so the Web client (e.g.: Web browser) can properly play a specific type of video. This stereotype was added because multimedia elements are a core feature of the RIAs [Busch and Koch, 2009].
2. *NormalClass*: a general purpose stereotype that can be used to specify some behaviours by attaching to it the following tagged values:
    - *memberIdentifier*: reference to an IDEPRIA attribute. This tagged value specifies which attribute contains the content that identifies a given instance of the *NormalClass*. This is very useful when choosing associations within the RIA, e.g.: when using a Web combo box. For instance, if a class *User* can have many *Product* instances associated, and the products should be displayed in a combo box, then, the attribute *name* of the class *Product* should be the *memberIdentifier* of the class *Product* to display the value contained in the attribute *name* in the Web combo box.
    - *memberPrivacy*: a stereotyped *NormalClass* can have zero, one or more *UserID* attributes. The designer can optionally pick one of them to be the attribute that defines the user who owns a given instance of the *NormalClass*. Then, if this tagged value is set, an authenticated user in the generated application will be able to modify, read or

delete only the instances that were previously created by the same authenticated user. This definition was added because it might be useful for RIAs that should have class instances with restricted user permissions.

- *layoutType*: a basic customization of the RIA that defines in how many columns should appear all the HTML form elements that will allow a given end-user to create or update a given instance of the class. The possibles values are defined in the *Class-LayoutType* enumeration: one, two or three columns of HTML form elements.
- *isManyToMany*: The many-to-many (M2M) cardinality refers to a relationship between two classes, in which any of the instances of one class can be associated to any of the instances of the other class, and vice-versa. This attribute should be set to true only if the class contains two IDEPRIA attributes that are tagged with *isForeignKey* equal to *true*. This attribute was added because the M2T software needs to know the M2M classes to create a specific RIA front-end interface so that the end-user can manage that relationship within the application.
- *generateCRUD*: If this tag is true, then, the M2T will generate a RIA interface that will allow to end-users the management of the instances of the class. Otherwise, only the persistent database schema for the class will be generated, excluding the Web services and Web user interfaces. This could be useful for RIAs that needs to store information in the database but without providing a particular Web interface to the end-user.

Additionally, Figure 4.2 shows the definition of the abstract stereotype *ReservedClass*, which extends from *NormalClass* and it is used as a generalisation of all the required stereotypes to design a fully working RIA. The following listings describe the required stereotypes that extend from *ReservedClass*, and each one of its corresponding required IDEPRIA attributes. Before beginning the description, it is important for the reader to know that the IDEPRIA attributes that are used in Figure 4.2, are defined in Figures 4.3, 4.4, 4.5, 4.6. However, the IDEPRIA attributes are going to be described now to make it easier for the reader to understand the relationship that each of them has with all the stereotypes that extend from *ReservedClass*.

1. *User*: stereotype to represent the users that will be able to log-in into the RIA and use the application as an authenticated user. These are the ten attributes of this stereotype, which are a reference to extensions of the *AttributeIDEPRIA* stereotype:
    - *userID*: the unique identifier of each *User* instance.

- *username*: the string username that the RIA end-user will use to log-in into the generated RIA.
- *password*: the string password that the RIA end-user will use to log-in into the generated RIA.
- *isAdmin*: the boolean that determines if the user instance is an administrator of the system or just a typical user. If a user instance has a true value in this attribute, then, that user will have no restrictions when executing Web services that might require special permissions.
- *wallpaper*: the URL that points to the wallpaper image that will appear in the RIA to the logged user. This attribute was added just so the end-users can customise a bit more the appearance of the RIAs.
- *stretchWallpaper*: the boolean that defines if the wallpaper image should be stretched or not before showing it to the logged user. This attribute was added just so the end-users can customise a bit more the appearance of the RIAs
- *isEnabled*: the boolean that determines if a given user is or not enabled to use the RIA. If an instance has a false value, then, that user instance will not be able to log-in into the RIA. This attribute was added because it could be very useful for a system administrator to restrict the access to the RIA to a particular end-user.
- *registrationDate*: the date in which the end-user entered to the system for the first time. This attribute was added because sometimes it could be useful to know when users start using the RIA.
- *theme*: the string name of the look and feel used that should be used for a particular user. Useful for users to customise the way the RIA looks by just changing this value. Currently, there is only one theme called *neptune*.
- *language*: the ISO 639-1 standard[1] code of the preferred language of the user. This attribute was added to provide to the users an easy way to change the preferred language once new languages are available in the system. Currently, for testing purposes, the two languages supported are: English (code: “en”) and Spanish (code: “es”).

2. *Person*: stereotype to represent the personal information of a given *User*. This stereotype was added to provide flexibility to future possibles users' personal information that could

[1] Standard ISO 639-2: `https://en.wikipedia.org/wiki/List_of_ISO_639-2_codes`

be required, so if a new user information is required in the future, then just the *Person* should be affected without modifying the *User*, which contains sensitive information like the password of the user. All of its three attributes are a reference to stereotyped IDEPRIA attributes that represents:

- *personID*: the unique identifier of each *Person* instance.
- *personName*: the string full name of the person.
- *personUser*: the system *User* that owns the information of this person.

3. *Profile*: stereotype to represent the different profiles that one of more users could have in the RIA. Different users can belong to different profiles, and each profile allows the possibility of execution of one or more actions. All of its two attributes are a reference to stereotyped IDEPRIA attributes that represents:
   - *profileID*: the unique identifier of each *Profile* instance.
   - *profileName*: the string name of the profile, so a given system administrator can assign a named profile to one or more users.
4. *Action*: stereotype to represent the Action that a given user can perform. An administrator user can assign a set of named actions to one or more profiles. Then, if a given user belongs to a given profile, that user will be able to run the actions enabled by that profile. All of its three attributes are a reference to stereotyped IDEPRIA attributes that represents:
   - *actionID*: the unique identifier of each *Action* instance.
   - *actionName*: the string name to identify a given action.
   - *actionModule*: the *Module* that will be affected when an user executes this action.
5. *UserProfile*: stereotype to represent the fact that a given user can belong to zero or more profiles and vice-versa. Only a system administrator user can make profile assignments to users. All of its two attributes are a reference to stereotyped IDEPRIA attributes that represents:
   - *upUser*: the *User* that belongs to the given *upProfile*.
   - *upProfile*: the *Profile* that will give the possibility to execute a set of actions to the *upUser*.
6. *ProfileAction*: stereotype to represent the that a given *Action* is allowed to be executed by users that belong to a given *Profile*. All of its two attributes are a reference to stereotyped

IDEPRIA attributes that represents:

- *paProfile*: the *Profile* that that now will permit users to execute the *paAction* action.
- *paAction*: the action that should be permitted to run by users that belongs to the profile *paProfile*.

7. *Module*: stereotype to represent the different modelled classes for a given RIA project. One module instance will be generated by the M2T software for each of the modelled classes, and each of these instances can be used generate the four different actions (CRUD operations) that can then be attached to one or more profiles.
   - *moduleID*: the unique identifier of each *Module* instance.
   - *moduleName*: the name of the module.
   - *desktopIcon*: the URL of a larger icon for the module.
   - *applicationIcon*: the URL of the smaller icon for the module.
   - *jsFile*: the URL of the JavaScript file that contains the front-end source code that will allow end-users manage the instances of a given module. This was added to allow loading of modules of the generated RIA in parts just when needed, without having to load all the modules at the beginning of the RIA and reducing starting latency.
   - *clientID*: the internal identification code used by the JavaScript front-end source code to launch a given module and display it to the end-user in the Web client.
8. *Shortcut*: stereotype to represent shortcuts of the modules that might or not be available in different sections of the RIAs for a given user. These shortcuts determine if a given module of the RIA will be accessible from two different places (desktop or quick launch). The M2T software can determine the definition of desktop and quick launch to generate the appropriate ways that will be available to the final user to interact with a given module.
   - *shortcutID*: the unique identifier of each *Shortcut* instance.
   - *isVisibleOnDesktop*: the boolean to determine if the module icon should be visible or not on the desktop of the RIA.
   - *isVisibleOnQL*: the boolean to determine if the module icon should be visible or not on the quick launch of the RIA.
   - *shortcutModule*: the *Module* manager that should be displayed to the user after clicking on the shortcut icon.

- *shortcutUser*: the *User* to which should be visible or not the module shortcut icon.

A.3 IDEPRIA Attributes

This section describes the stereotypes that extend from the UML meta-class *Property*, which are shown in Figure 4.3. The abstract stereotype *AttributeIDEPRIA*, represents a simple extension of a UML property to which it is possible to attach the following tagged values:

1. *attributeDatatype*: data type of the attribute. The possibles values are defined in the enumeration *AttributeDatatypes*. These type literals were obtained from the MySQL website[2], and the same restrictions described on each type will be applied. This will allow the persistent storage of full instances in database engines like MySQL.

2. *isFileUploadPath*: boolean property that determines if the RIA form should display this field as an HTML file picker that will permit a user to upload files. Then, if a file is submitted, it will be locally stored on the back-end Web server, and this attribute will store the URL that will permit the access to the file. This file upload feature is very useful for file-management-oriented RIAs.

3. *isManyToManyOwner*: when a class has the tagged value *isManyToMany* equals to *true*, then it is possible to define if only one of attributes owns the other one, or if they both owns each other. This boolean is useful for the M2T software so it is possible to know which Web services and front-end functionalities should be provided to the end-user. If the two members, A and B, are marked as *manyToManyOwners*, then, the management Web services that will be generated are CRUD operations of B instances for any given A instance and vice-versa.

4. *isModificationDate*: boolean that represents that this attribute will have the current date after an instance is created or after the modification of any of the attributes of the instance. Then, that instance will always have the last modification date of the instance in this attribute.

5. *isCreationDate*: boolean that represents that when an instance is created, then, that instance will automatically have the current date in this attribute.

6. *allowNull*: boolean that determines if an instance of this attribute will be allowed to have the null (empty) value.

[2] MySQL data types: `https://dev.mysql.com/doc/refman/5.7/en/data-types.html`

7. *isPrimaryKey*: boolean that determines if this attribute will be the primary key of the class, which then will be used for many purposes, including the definition of the primary key column of the class table.

8. *isForeignKey*: boolean that determines if this attribute is a foreign key from another instance. This will be used to allow an end-user to choose which attribute instance of the foreign class should be associated with the current class.

9. *isAutoincrement*: boolean that determines if the value of this attribute is going to be automatically assigned by the database engine by auto-incrementing the value.

10. *isUnique*: boolean that establishes that two different instances of the same class could not have the same value in this attribute.

11. *attributeID*: integer identifier of this attribute.

12. *attributeLength*: integer value that defines the size of the number data types (*Integer, TinyInteger, SmallInteger, BigInteger, DecimalNumber, MoneyNumber*) and the length of the string data types (*Char, VarChar*) shown in the *AttributeDatatypes*.

13. *attributeDecimals*: number of digits that are going to be considered as decimals of the number. This stereotype only applies to attributes that have the *attributeDatatype* equals to *DecimalNumber* or *MoneyNumber*.

14. *attributeDefaultValue*: this is a string value that will be used as default value for this attribute, in case that no value has been provided to it when creating the instance. If the data type of this attribute is not string, then, a casting to the proper data type will be tried.

Figures 4.4, 4.5, 4.6 define extensions to the already described stereotype *AttributeIDEPRIA*. Mostly, these stereotypes are the ones used as tagged values attached to the stereotypes that extend from *ReservedClass* stereotype, which were already described when explaining the *ReservedClass* stereotype. The following listing provides the name of all of them:

- *ActionID*, *ActionName*, which extend from the abstract stereotype *ActionAttribute*.
- *ShortcutID*, *isVisibleOnDesktop*, *isVisibleOnQuickLaunch*, which extend from the abstract stereotype *ShortcutAttribute*.
- *ProfileID*, *ProfileName*, which extend from the abstract stereotype *ProfileAttribute*.
- *UserID*, *UserName*, *UserPassword*, *UserWallpaper*, *UserStretchWallpaper*, *UserIsEnabled*, *UserRegistrationDate*, *UserTheme*, *UserLanguage*, *UserIsadmin*, which extend from the

abstract stereotype *UserAttribute*.

- *PersonID*, *PersonName*, which extend from the abstract stereotype *PersonAttribute*.
- *ModuleID*, *ModuleName*, *ModuleDesktopIcon*, *ModuleApplicationIcon*, *ModuleJSFile*, *ModuleClientID*, which extend from the abstract stereotype *ModuleAttribute*.

Additionally, the *UserID* stereotype, shown in Figure 4.5, has two tagged values useful to define the following specific behaviours:

1. *isUserCreator*: this represents that the *UserID* attribute is an attribute of a class that will always have as value the reference to the currently authenticated user of the RIA. This stereotype is very useful for RIAs that want to keep track of which user created a given instance.
2. *isUserModifier*: similar to the *isUserCreator*, but now this attribute will keep track of the last authenticated user that modified a given instance.

Appendix B

Example of a M2T configuration file

```
{
  "id": 23,
  "shortname": "pyextjs",
  "name": "Python back-end + ExtJS front-end + MySQL",
  "description": "Generates the RIA in a compressed .zip file",

  "static": [{
    "type": "folder", "src": "static/", "dst": "static/"
  }],

  "templates": [{
    "src": "Login.jinja2", "dst": "Login.py"
  }],

  "permodule": [{
    "tpl": "model.jinja2",
    "dst": "models/$modname$.py",
    "options": [{"name":"capitalize"}]
  }]
}
```

Listing B.1: Example of a M2T configuration file.

Appendix C

Example of a JINJA2 template

```
1 CREATE SCHEMA IF NOT EXISTS `{{ toLower(project.name) }}` DEFAULT CHARACTER
    SET utf8 COLLATE utf8_general_ci ;
2 USE `{{ toLower(project.name) }}` ;
3
4 -- ------------- --
5 -- FOREIGN KEYS  --
6 -- ------------- --
7
8 {% for module in project.modules %}{% if module.type != "HTML5 Video" %}{%
    for member in module.members %}{% for relation in member.relations
    %}ALTER TABLE `{{ toLower(relation.memberAsoc.module.name) }}` ADD
    CONSTRAINT `{{ toLower(relation.getFkName()) }}` FOREIGN KEY (`{{
    relation.memberAsoc.name }}`) REFERENCES `{{
    toLower(relation.memberOwner.module.name) }}` (`{{
    relation.memberOwner.name }}`) ON DELETE CASCADE ON UPDATE CASCADE;
9 {% endfor %}{% endfor %}{% endif %}{% endfor %}
```

Listing C.1: MySQL Database schema - JINJA2 template

Appendix D

Consent form for the Think-Aloud sessions

Consent Form

Project: A Model-Driven Development environment for rapid prototyping of Rich Internet Applications

Researcher: Victor Cajes Gonzalez
University Tutor: Perdita Stevens

Researcher address: 11 Ascham Court, EH8 9LQ, Edinburgh
Email: s1600253@sms.ed.ac.uk **Mobile number:** +447460670388

If you agree, please tick all the following boxes:

1. I agree to take part in the above study. The study includes: ☐
   a. Being a participant of a Think-Aloud session.
   b. Being interviewed about the Think-Aloud session.
   c. Filling a questionnaire at the end of the interview.

2. I understand that taking part is my choice. ☐
   I do not have to take part and I can stop at any time.
   I understand that I do not have to give a reason for changing my mind.
   If I stop doing the study, it will not affect my legal rights.

3. I understand that all the information collected will be written in the researcher's master thesis book and other possible reports/publications. ☐
   Also, reports and publications will not use my real name.
   No one will know who I am from the information in reports.

4. I understand and consent to my Think-Aloud session and interview being audio-recorded and video-recorded. ☐

5. I understand that direct, anonymised quotes from my interview may be used to illustrate findings in reports or publications. ☐

6. I understand that sections of data collected during the study may be looked at by responsible individuals from the University of Edinburgh or from regulatory authorities, where it is relevant to my taking part in the study. I give permission for these people to have access to this data. ☐

7. **I agree to take part in the study.** ☐

_____ _____

Name of the participant **Signature of the participant**

Appendix E

Think-Aloud Task definitions

E.1 Task 1

Create one class named “Users”, which should have the following attributes:

1. **id**: an integer that should be incrementally auto-allocated by the system. This is the attribute that uniquely identifies an user of the system.
2. **name**: a variable length characters. The maximum possible number of characters are 80. This is the name of the user.

Create one class named “Files”, which should have the following attributes:

1. **id**: an integer that should be incrementally auto-allocated by the system. This is the attribute that uniquely identifies a file uploaded by the user.
2. **filepath**: a string that represents the file path of the uploaded file by the currently logged-in user. The maximum possible number of characters are 255.

Create one association named “uploader” in the class “Files”, which should reference to the **id** attribute of the **Users** class. In other words, the class “Files” should have an integer attribute named “uploader” that is a Foreign Key that references to the **id** attribute of the **Users** class. Also, when a given logged-in authenticated user creates one “Files” instance in the system, the value of this attribute should be automatically set by the system to the **id** of the current authenticated user.

E.2 Task 2

This task ask you to design and develop a prototype of a Rich Internet Application using the IDEPRIA tool. This application should allow users to join to courses, and also, to submit assignments to those courses. The Rich Internet Application should meet the following requirements:

1. All the users of the system should be able to authenticate with their username and password before they can start using the system. Also, it should be the possibility to identify which users of the system are administrators and which ones are not.
2. It should be possible to store the following personal information for each of the users: name, email, personal document number, address, phone number.
3. There should be the possibility to create/modify courses that have a string code of 10 fixed characters that uniquely identifies them, a short string name, and a level of difficulty represented as an integer number between 1 and 10. Observations: these courses are always active, and are not date/time dependent, so any user can join/leave the course at any moment. Also, there is not need to keep track of the specific date-time when an user joins to or leaves from a given course.
4. The users should be able to submit assignments, which consists on an auto-increment integer identifier, a title for the submission, a long text where the user can input the details of the submission, a date-time of the submission and one string field with a maximum of 255 characters that will store the URL to one optional file that the user can upload with the submission.
5. When an authenticated user makes a submission, the system automatically should save the identifier of the user that made the submission and should set the current time as the submission date-time.
6. Finally, the system should provide a video player that shows a video that explains how to use the system. The video was already made, its format is *mp4* and the URL link is: `http://www.sample-videos.com/video/mp4/480/big_buck_bunny_480p_1mb.mp4`

E.3 Task 3

This task ask you to use the IDEPRIA tool to generate and download the ready-to-deploy source code of the Rich Internet Application that you have designed.

Appendix F

Case-study questionnaire

IDEPRIA Evaluation Questionnaire

Consent statement: This questionnaire is intended for computer software developers only. All the answers will be used in a dissertation project to be presented at the University of Edinburgh. The answers might be published publicly. Please do not provide any information that can be used to identify yourself. Feel free to not answer questions if do not want to.

1. What is your age in years?

   ◯ 0 - 18

   ◯ 19 – 35

   ◯ 36 – 54

   ◯ 55+

2. Have you already obtained an informatics-related degree?

   ◯ No

   ◯ Yes

3. Have you ever used a software tool to design an UML Class Diagram?

   ◯ No

   ◯ Yes. Please write the name of the tool if you remember it.

   _____

4. Have you ever followed the Model-Driven Development approach to develop any kind of software? This approach allows you to obtain the source code of the software from the models that you have designed.

   ◯ No

   ◯ Yes

5. How difficult were to you in average the definition of the tasks that you were asked to do in the Think-Aloud session?

   ◯ Very difficult

   ◯ Difficult

   ◯ Neutral

   ◯ Easy

   ◯ Very easy

6. Please rate if you agree or disagree with the following statements.

General impressions	**strongly disagree**	**disagree**	**uncertain**	**agree**	**strongly agree**
The usage of this tool was frustrating.	☐	☐	☐	☐	☐
My expectations for the tool were fulfilled.	☐	☐	☐	☐	☐

Model development	**strongly disagree**	**disagree**	**uncertain**	**agree**	**strongly agree**
Developing a model by using this tool was easy.	☐	☐	☐	☐	☐
Developing a model by applying this tool was successful.	☐	☐	☐	☐	☐
I was able to develop the given scenario completely.	☐	☐	☐	☐	☐
I was able to develop the given scenario accurately.	☐	☐	☐	☐	☐
It was frustrating that I could not model some UML class diagram elements. E.g.: multiplicity of associations, abstractions.	☐	☐	☐	☐	☐
The tool must provide a way to model all the UML class diagram elements.	☐	☐	☐	☐	☐

Model-Driven Development	**strongly disagree**	**disagree**	**uncertain**	**agree**	**strongly agree**
The tool permits the modelling of the most important elements of a prototype of a Web Application (WebApp).	☐	☐	☐	☐	☐
I was able to customize the WebApp as I wanted.	☐	☐	☐	☐	☐
The automatically-generated WebApp was enough for a “prototype” of the application that I wanted.	☐	☐	☐	☐	☐
The automatically-generated WebApp prototype was what I modelled.	☐	☐	☐	☐	☐
The automatically-generated WebApp prototype was what I needed.	☐	☐	☐	☐	☐

7. How familiar was to you the diagram and modelling elements that you were able to use and customise while performing the Think-Aloud session?

not at all familiar	**slightly familiar**	**somewhat familiar**	**moderately familiar**	**extremely familiar**
☐	☐	☐	☐	☐

8. How difficult was for you to learn and use on-the-fly the tool that required you only one extended UML Class Diagram?

very difficult	**difficult**	**uncertain**	**easy**	**very easy**
☐	☐	☐	☐	☐

9. The IDEPRIA tool allowed you to generate the source code of a prototype of the Web Application from what you have modelled in just one extended UML Class diagram. To what extent do you agree that these elements are enough to model a prototype of a Web Application that you might need to develop in the future?

strongly disagree	**disagree**	**uncertain**	**agree**	**strongly agree**
☐	☐	☐	☐	☐

10. To what extent do you agree that other types of diagrams, different than a UML Class diagram or extensions of it, are as well necessary to be able to model other important aspects that you might require for a prototype of a Web Application?

strongly disagree	**disagree**	**uncertain**	**agree**	**strongly agree**
☐	☐	☐	☐	☐

11. Instead of having to learn and design just one extended (and possibly familiar) UML Class Diagram, how desirable would be for you to have to learn more types of diagrams individually, then learn how to link/connect all the elements between the diagrams, and then having to actually model all those multiple diagrams to design just a prototype of the Web Application that you want?

very undesirable	**undesirable**	**neutral**	**desirable**	**very desirable**
☐	☐	☐	☐	☐

12. What would you add/modify/remove from the designing elements to make it a more suitable tool for modelling and developing just prototypes of Web Applications?

_____

_____

_____

_____

Bibliography

[Albert et al., 2011] Albert, M., Cabot, J., Gómez, C., and Pelechano, V. (2011). Generating operation specifications from uml class diagrams: A model transformation approach. *Data Knowl. Eng.*, 70(4):365–389.

[Ambler and Lines, 2012] Ambler, S. and Lines, M. (2012). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press, first edition.

[Baerisch, 2010] Baerisch, S. (2010). *Domain-Specific Model-Driven Testing*. Wiesbaden Vieweg+Teubner.

[Bernardi et al., 2014] Bernardi, M. L., Lucca, G. A. D., and Distante, D. (2014). Model-driven fast prototyping of rias: From conceptual models to running applications. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 250–258.

[Bin Ahmad and Iahad, 2013] Bin Ahmad, M. A. and Iahad, N. A. (2013). Websites usability instrument validation using think-aloud method. In *International Conference of Information and Communication Technology*, pages 208–212. IEEE.

[Booch, 2005] Booch, G. (2005). *The unified modeling language user guide*, pages 7–15. Pearson Education India, first edition.

[Bozzon et al., 2006] Bozzon, A., Comai, S., Fraternali, P., and Carughi, G. T. (2006). Conceptual modeling and code generation for rich internet applications. In *Proceedings of the 6th International Conference on Web Engineering*, ICWE '06, pages 353–360, New York, NY, USA. ACM.

[Breeding, 2012] Breeding, M. (2012). *Cloud computing for libraries*, volume 11, pages 109–110. American Library Association.

[Busch and Koch, 2009] Busch, M. and Koch, N. (2009). Rich internet applications: State-of-the-art. Technical report, Ludwig-Maximilians-Universität München.

[Dobing and Parsons, 2006] Dobing, B. and Parsons, J. (2006). How uml is used. *Commun. ACM*, 49(5):109–113.

[Erikson and Simon, 1984] Erikson, K. and Simon, H. (1984). *Protocol analysis : verbal reports as data*. Bradford books. MIT, Cambridge, Mass. ; London.

[Fowler and Scott, 2000] Fowler, M. and Scott, K. (2000). *UML distilled: a brief guide to the standard object modeling language*, pages 1–9. Addison Wesley, Boston, MA, USA, second edition.

[Fraternali et al., 2010] Fraternali, P., Rossi, G., and Sánchez-Figueroa, F. (2010). Rich internet applications. *IEEE Internet Computing*, 14(3):9–12.

[Fuentes and Vallecillo, 2004] Fuentes, L. and Vallecillo, A. (2004). An introduction to uml profiles. *The European Journal for the Informatics Professional*, 5(2):6–13.

[Golobisky and Vecchietti, 2005] Golobisky, M. F. and Vecchietti, A. (2005). Mapping uml class diagrams into object-relational schemas. In *Proceedings of Argentine Symposium on Software Engineering*, pages 65–79.

[Jeon and Lee, 2007] Jeon, J. and Lee, S. (2007). Toward a mobile rich web application – mobile ajax and mobile web 2.0. Technical report, World Wide Web Consortium (W3C).

[Kapteijns et al., 2009] Kapteijns, T., Jansen, S., Brinkkemper, S., Houët, H., and Barendse, R. (2009). A comparative case study of model driven development vs traditional development: the tortoise or the hare. *From code centric to model centric software engineering: Practices, Implications and ROI*, 22.

[Koffka, 1935] Koffka, K. (1935). *Principles of Gestalt Psychology*. Routledge & Kegan Paul PLC.

[Krogmann and Becker, 2007] Krogmann, K. and Becker, S. (2007). A case study on model-driven and conventional software development: The palladio editor. In *Software Engineerings*, volume 106, pages 169–176. Citeseer.

[Marco Brambilla, 2017] Marco Brambilla, Jordi Cabot, M. W. (2017). *Model-Driven Software Engineering in Practice*. Morgan and Claypool Publishers, second edition.

[Martínez et al., 2012] Martínez, Y., Cachero, C., and Meliá, S. (2012). *Evaluating the Impact of a Model-Driven Web Engineering Approach on the Productivity and the Satisfaction of Software Development Teams*, pages 223–237. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Martínez et al., 2013] Martínez, Y., Cachero, C., and Meliá, S. (2013). Mdd vs. traditional software development: A practitioner's subjective perspective. *Inf. Softw. Technol.*, 55(2):189–200.

[Meliá et al., 2008] Meliá, S., Gómez, J., Pérez, S., and Díaz, O. (2008). A model-driven development for gwt-based rich internet applications with ooh4ria. In *8th International Conference on Web Engineering*, pages 13–23.

[Moreno et al., 2008] Moreno, N., Romero, J. R., and Vallecillo, A. (2008). *An Overview Of Model-Driven Web Engineering and the Mda*, pages 353–382. Springer London, London.

[Mou and Ratiu, 2012] Mou, D. and Ratiu, D. (2012). Binding requirements and component architecture by using model-based test-driven development. In *2012 First IEEE International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*, pages 27–30.

[Nielsen, 1993] Nielsen, J. (1993). *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[O. M. G., 2004] O. M. G., U. (2004). 2.0 superstructure specification. Technical report, OMG.

[Orban, 2011] Orban, D. (2011). Templating and automatic code generation for performance with python. Technical report, Groupe d'études et de recherche en analyse des décisions and the Department of Mathematics and Industrial Engineering, École Polytechnique, Montréal, QC, Canada.

[Phillips, 2014] Phillips, A. (2014). *The Usefulness of 'Think-Aloud' for Evaluating Questionnaires in use in the Health Domain*. PhD thesis, The University of Manchester, Manchester, UK.

[Red et al., 2014] Red, E., French, D., Hepworth, A., Jensen, G., and Stone, B. (2014). Multi-user computer-aided design and engineering software applications. In *Cloud-Based Design and Manufacturing (CBDM): A Service-Oriented Product Development Paradigm for the 21st Century*, pages 25–62. Springer International Publishing.

[Runeson and Höst, 2008] Runeson, P. and Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131.

[Runeson et al., 2012] Runeson, P., Host, M., Rainer, A., and Regnell, B. (2012). *Case Study Research in Software Engineering*. John Wiley and Sons, first edition.

[Schalles, 2013] Schalles, C. (2013). *Usability evaluation of graphical modeling languages an empirical research study*. Springer Gabler, Wiesbaden.

[Selic, 2003] Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25.

[Stefano et al., 2010] Stefano, F., Borsci, S., and Stamerra, G. (2010). Web usability evaluation with screen reader users: implementation of the partial concurrent thinking aloud technique. *Cognitive Processing*, 11(3):263–272.

[Stevens, 2006] Stevens, P. (2006). *Using UML: software engineering with objects and components*, pages 45–46. Pearson Education, second edition.

[Wallace et al., 2002] Wallace, C., Cook, C., Summet, J., and Burnett, M. (2002). Assertions in end-user software engineering: a think-aloud study. In *2002 IEEE Computer Society International Symposium*, pages 63–65, USA. IEEE.