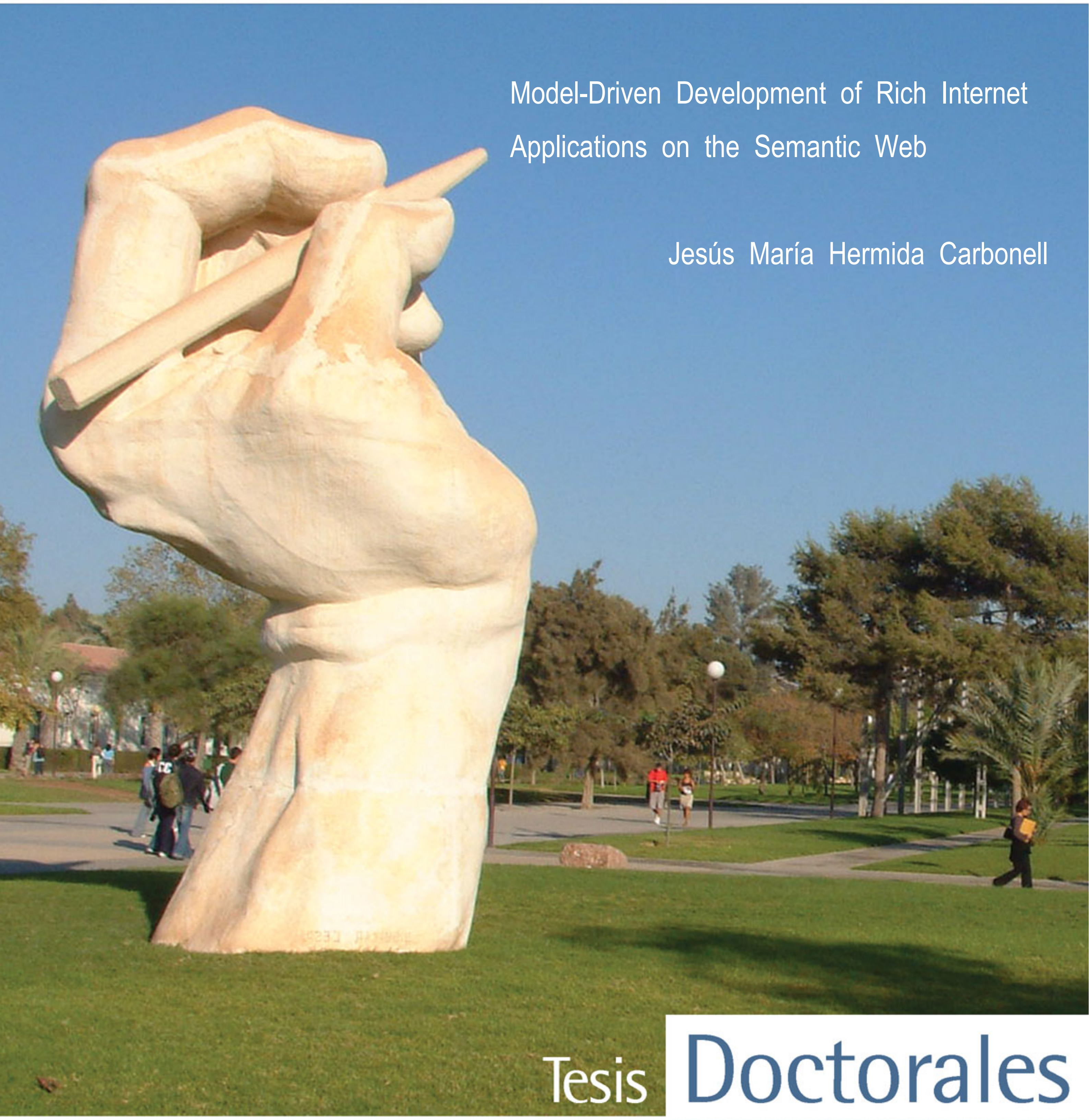


Model-Driven Development of Rich Internet Applications on the Semantic Web

Jesús María Hermida Carbonell

Universitat d'Alacant
Universidad de Alicante

Model-Driven Development of Rich Internet Applications on the Semantic Web

PhD Thesis

Jesús M. Hermida Carbonell

2013

Universitat d'Alacant
Universidad de Alicante

Model-Driven Development of Rich Internet Applications on the Semantic Web

Dissertation

Presented to the Department of Software and Computing Systems, University of Alicante, in partial fulfilment of the requirements for the title of

Doctor of Philosophy

Candidate: Jesús María Hermida Carbonell

Advisors: Dr. Santiago Luís Meliá Beigbeder
Dr. Juan Andrés Montoyo Guijarro

April 2013

ACKNOWLEDGEMENTS

“Success is not final, failure is not fatal: it is the courage to continue that counts.” WINSTON CHURCHILL

I would sincerely like to thank all those who have been by my side and have lent me a hand when I needed it.

I would like to express my gratitude to my supervisors Prof. Dr. Santiago Meliá and Prof. Dr. Andrés Montoyo for all their support in the difficult moments. Many thanks to Prof. Dr. Santiago Meliá for all the discussions about “semantics”, models, colourful transformations and software architectures (in particular, the *Model-View-ViewModel* pattern). Many thanks to Prof. Dr. Andrés Montoyo for showing me the pathway to different fields: Semantic Web, ontology engineering and software models; and for all his support during these years as a student, from the first weekly meetings to the last day.

I would like to thank Prof. Dr. Jaime Gómez for all his support as leader of the Web Engineering group (IWAD) and as supervisor of this thesis during the first part of this path. Many thanks to him for patiently listening to my firsts ideas of model-driven methodologies.

I would also like to thank all the members of the Natural Language Processing group (GPLSI) for their friendship. In particular, I would like to thank Prof. Dr. Manuel Palomar for making me feel welcomed in the group. From this period, I especially remember Prof. Dr. Rafael Muñoz, for our discussions about information extraction on notary documents, and Prof. Dr. Mayte Romá, for our long discussions about ontologies and knowledge representation, which have made me consider all the problems from different viewpoints.

I would like to thank all those have shared an office with me: Héctor, Elena, Rubén, Jorge, Yoan, Dianelys, José Manuel, Javi and many others.

I would like to express my gratitude to José Javier and Sergio for their assistance and guidance with the Eclipse Modelling Framework and the OIDE CASE tool.

I would also like to thank all the members of the Organisations, Information and Knowledge group at the University of Sheffield for having welcomed me to be part of their team during my internship there. I would especially extend my gratitude to its director, Prof. Dr. Fabio Ciravegna, and all my colleagues and friends: Deep, Rodrigo, Jonathan, Anna Lisa, Elisabeth, Greg and Mathew.

I would also like to thank all the members of the Digital Earth and Reference Data unit (Institute for Environment and Sustainability) at the European Commision Joint Research Center in Ispra for their help. Especially, to the unit head, Dr. Alessandro Annoni, and the members of the SHAPE action, led by Dr. Paul Smits, and my supervisor, Dr. Michael Luzt. Thanks to Chris, Andrej, Vlado, Tomas, Michalis, Ángel, Robert and all those who work with me every day.

I would not like to forget all the help and support from my parents, M. Carmen and Jesús, who have been by my side during the difficult moments and have encouraged me several times to arrive to the finish line. Also thanks to my sister, Carmen, who always lent me a hand when I needed it, and all my family members and friends.

I would like to especially thank Alexandra for all her help and for the energy and confidence she gives me every day along this path we are walking together. This path started with some endless meetings long ago. Thank you for cheering me up in the moments when the finish line seemed to get further away.

Finally, I would like to thank the Spanish Ministry of Education for their support during the period 2008-2012 by means of the FPU scholarship I was granted (code AP2007-03076), which allowed me to complete this doctoral thesis at the University of Alicante and all the training needed to become a researcher and a lecturer in the near future.

AGRADECIMIENTOS

"El éxito no es definitivo, el fracaso tampoco: es el coraje de continuar lo que cuenta". WINSTON CHURCHILL

Me gustaría agradecer de corazón a todos aquellos que me han acompañado durante este largo camino y me han dado energías en los momentos que más lo necesitaba.

Agradecer a mis directores Santiago Meliá y Andrés Montoyo por toda su ayuda en los momentos complicados. A Santiago Meliá, por todas las discusiones sobre "semántica", modelos, transformaciones variopintas y arquitecturas software (en especial, el patrón *Model-View-Viewmodel*). A Andrés Montoyo, por introducirme en el mundo de la Web Semántica, las ontologías y los modelos, y por toda la ayuda recibida durante todos los años como doctorando, desde las primeras reuniones semanales los viernes hasta el último día.

A Jaime Gómez, por todo el apoyo brindado como director del grupo de investigación Ingeniería Web, Aplicaciones y Desarrollos (IWAD) y como director de tesis en la primera parte de este camino. Gracias por escuchar pacientemente mis primeras ideas para el desarrollo de una metodología dirigida por modelos.

Agradecer a todos los miembros del grupo de investigación en Procesamiento del Lenguaje y Sistemas de Información (GPLSI) por todo su cariño en el día a día. Agradezco a su director, Manuel Palomar, por la siempre cálida acogida dentro del grupo. De este periodo, me llevo un especial recuerdo de Rafael Muñoz, por aquel verano notarial revisando sujetos y objetos, y de Mayte Romá, por las largas discusiones sobre ontologías y representación de conocimiento, que me han ayudado a intentar ver día a día las cosas desde otras perspectivas. A todos los que han compartido despacho(s) conmigo: Héctor, Elena, Rubén, Jorge, Yoan, Dianelys, José Manuel, Javi y a otros muchos.

Expresar mi agradecimiento a Jose Javier y a Sergio su asistencia y su guía en el uso del Eclipse Modelling Framework y la herramienta OIDE.

Agradecer también a todos los miembros del grupo *Organisations, Information and Knowledge* (OAK) de la Universidad de Sheffield por su grata acogida durante los meses de estancia allí. En especial a su director, Fabio Ciravegna, y a mis compañeros: Deep, Rodrigo, Jonathan, Anna Lisa, Elisabeth, Greg y Mathew.

Agradecer a todos los miembros de la unidad *Digital Earth and Reference Data* (*Institute for Environment and Sustainability*) del *European Commission Joint Research Center* en Ispra por el afecto recibido desde el primer día de meses en Italia. En especial, al líder de unidad, Alessandro Annoni; a los miembros de la acción SHAPE, liderada por Paul Smits; y a mi responsable, Michael Luzt. Gracias a Chris, Andrej, Vlado, Tomas, Michalis, Ángel, Robert y a todos los que trabajan conmigo día a día.

Me gustaría no olvidarme de toda la ayuda, apoyo y cariño de mis padres, Mª Carmen y Jesús, durante estos años, que han estado a mi lado en momentos difíciles y me han estimulado en muchas ocasiones para poder llegar a la meta. Extender este agradecimiento a mi hermana Carmen, que me ha dado una mano cuando me ha hecho falta, y al resto de mi familia y amigos.

Agradecer de forma muy especial a Alexandra por toda la ayuda y por él ánimo, apoyo y confianza que me transmite en el día a día de este camino que estamos andando juntos. Camino que empezó con aquellas reuniones infinitas hace mucho. Gracias por seguir animándome en los momentos en que la meta parece alejarse.

Sólo me queda terminar agradeciendo al Ministerio de Educación del Gobierno de España por la ayuda recibida durante el periodo 2008-2012 a través de la beca FPU (Formación de Profesorado Universitario) código AP2007-03076 , que me ha permitido realizar esta tesis doctoral y formarme de forma integral como investigador y profesor universitario dentro de la Universidad de Alicante.

ABSTRACT

In the last decade, the Web 2.0 brought technological changes in the manner of interaction and communication between users and applications, and among applications as well. Rich Internet Applications (RIA) offer user interfaces with a higher level of interactivity, similar to desktop interfaces, embed multimedia contents and minimise the communication between client and server components. Nonetheless, RIAs behave as black boxes that show the information in a user-friendly manner but this information can be only visualised gradually, according to the events triggered by the users on the Web browser, which limits the access of software agents, e.g., Web searchers.

In the context of the present Internet, where the value has been moved from the Web applications to the data they manage, the use of open technological solutions is a need. In this way, the Semantic Web was aimed at solving issues of semantic incompatibility among systems by means of standard techniques and technologies (from knowledge representation and sharing to trust and security), which can be the key to solving the issues detected in RIA.

Although some solutions exist, they do not cover all the possible types of RIA or they are dependent on the technology chosen for the implementation of the Web application. As a first contribution, this thesis introduces the concept of Semantic Rich Internet Application (SRIA), which can be defined as a RIA that extensively uses Semantic Web technologies to provide a representation of its contents and to reuse existing knowledge sources on the Web. The solution proposed is adapted to the existing RIA types and technologies. The thesis presents the architecture proposed for this type of application, describing its

software modules and components. The evaluation of the solution was performed based on a collection of case studies.

The development of Web applications, especially in the context of the Semantic Web, is a process traditionally performed manually and, given the complexity of the SRIA applications in this case, it is a process which might be prone to errors. The application of model-driven engineering techniques can reduce the cost of development and maintenance (in terms of time and resources) of the proposed applications, as demonstrated their use in other types of Web applications. Moreover, they can facilitate the adoption of the solution by the community.

In the light of these issues, as a second contribution, this thesis presents the S^{m}4RIA methodology (Semantic Models for RIA) for the development of SRIA, as an extension of the OOH4RIA methodology. The thesis describes the development process, the models (with the corresponding metamodels) and the transformations included in the methodology. The evaluation of the methodology consisted in the development of the case studies proposed. The application of this model-driven methodology can speed up the development of these Web applications and simplify the reuse of external sources of knowledge.

Finally, the thesis describes the *S^{m}4RIA extension for OIDE*, i.e., an extension of the OIDE CASE tool that implements all the elements of the S^{m}4RIA methodology.

TABLE OF CONTENTS

INDEX OF TABLES

INDEX OF FIGURES

Chapter 1. INTRODUCTION

In the present knowledge society, the need to access information has grown exponentially. During the past two decades, the Internet has experienced a continuous, relatively rapid, evolution from several points of view all intended to fulfil the end user's information requirements. This led to the creation of different trends, oriented to a specific subset of these requirements. Murugesan (Murugesan, 2008) identified and described some of them: Web 1.0, Web 2.0, Rich Internet Applications, the Semantic Web or the Mobile Web (even though there could be others).

Among these trends, this thesis is focused on the branch of Rich Internet Applications. These applications were introduced by the Web 2.0, together with the social applications. Their interfaces provided functionalities that up to that moment had been considered only in desktop interfaces. Based on technologies such as Flex, Silverlight or AJAX, i.e., HTML(5) and JavaScript, among others, these applications include user interfaces with high degrees of interactivity and dynamicity, embedding multimedia elements, which can retrieve data from the Web server without having to change the Web page or click on a link or button.

An issue of data access, sharing and interoperability.

Despite these advantages regarding data visualisation, Rich Internet Applications have an important drawback: the existing Web search engines, which for several users are the main entry point to the Web contents, cannot easily crawl and index the information they manage. As a result, Web users cannot easily find the content shown, which can at the same time prevent developers and enterprises from creating and using this type of applications for their business.

The behaviour of RIA interfaces is driven by user events, i.e., their information is shown based on the users' demands, expressed by manually triggering certain events on the user interface components. This event-driven behaviour makes the access to the data complicated independently from the technology chosen to implement the RIA. In this scenario, HTML-based RIAs have an advantage over plug-in-oriented RIAs, implemented with other technologies such as Silverlight or Flex, since their contents are shown using textual representations embedded in HTML code, similar to the traditional HTML interfaces of the Web 1.0. Some technology-dependent solutions (e.g., for Flex or Silverlight) are available, which enable the sharing of a part of the content. Those technology-independent solutions for crawling RIA are expensive in terms of time and resources, and cannot access all the data they show.

Reusing efforts on the Web.

Given the maturity of the Internet and the technologies developed, the access to the data stored in RIAs for different user types could be facilitated by reusing the efforts spent in other trends of the Web. In this case, the attention is on the techniques and technologies for knowledge management developed in the context of the Semantic Web. The Semantic Web (Berners-Lee et al., 2001) considers software systems as first-class users that help human users with their tasks, not only as mere tools for managing and visualising information, but also helping in the whole process of acquisition, management and decision-making. To this aim, new technologies and tools have been created to provide explicit, disambiguated meaning to the information flowing throughout the Web using techniques for knowledge capturing, representation and management. Understanding the information contained in Web sites leads to a higher degree of interoperability (at three levels: lexical, syntactic and semantic) among software components on the Web. From this perspective, the problems related to the access to RIA data can be restated as a problem of lack of interoperability, which could be addressed with Semantic Web techniques and technologies.

Development challenges and model-driven engineering.

The combination of these non-complementary Web trends also involves several trade-offs and challenges in the engineering of Web applications. As Murugesan stated (Murugesan, 2008), all these could be summarised into a main challenge: "to design and develop sustainable Web systems for better *a)* usability, interface design, and navigation; *b)* comprehension; *c)* performance (responsiveness); *d)* security and integrity; *e)* evolution, growth, and maintainability; *f)* testability; and *g)* mobility."

The success of the solution proposed for RIA and its acceptance will directly depend on the cost of addressing these challenges, which is usually high in terms of resources and time, due to the complexity of the functionalities required by these applications and the dynamics of the Web scenario. In this last decade, several model-driven methodologies for engineering Web applications have appeared, dealing with the challenges mentioned and facilitating the processes to develop complex Web applications. Model-driven methodologies propose development processes whose activities are oriented to the design of software models. Moreover, they can define collections of transformations to obtain new models from existing ones or the final software modules of the target application. Model-driven methodologies are especially adapted to develop Web systems that provide better usability and facilitate their evolution and maintainability once the development has been concluded (aspects *a)* and *f*) from Murugesan's definition). This type of development techniques, together with a CASE[1] tool that supports it, can reduce the costs related to the mentioned aspects in complex Web applications.

Nevertheless, none of the existing model-driven Web engineering methodologies effectively combines the elements required for the development of a solution to deal with the aforementioned issues of data access and sharing in RIA using Semantic Web technologies. Existing methodologies (e.g., WebML) contain part of the elements needed (e.g., development of rich user interfaces, ontologies, access to Web services), but these elements remain unconnected. In addition, they

[1] Computer-Aided Software Engineering

are not completely aligned to the new processes for knowledge sharing and reuse on the Semantic Web such as the ones introduced by the Web of Data.

1.1 THE OBJECTIVES OF THIS THESIS

In the light of the issues identified, the research conducted in this thesis intends to answer three research questions:

RQ1 *– Is it possible to improve the interoperability of Rich Internet Applications with other software systems (such as, Web search engines) using existing techniques, technologies and resources from the Semantic Web?*

RQ2 *– How can the existing model-driven methodologies be extended in order to develop the solution to the problems detected in Rich Internet Applications?*

RQ3 *– How can the proposed solutions be implemented in a CASE tool?*

In order to answer these questions, several objectives have been proposed. They are described and motivated as follows:

Objective 1) *Improve the interoperability of Rich Internet Applications with text-driven software systems on the Web (e.g., searchers).*

O1.1) Improve the exportability of the data offered by Rich Internet Applications.

O1.2) Improve the access to information related to multimedia elements.

O1.3) Combine techniques, technologies and resources already developed in the Semantic Web with Rich Internet Applications technologies.

O1.4) Develop a collection of use cases that assess the validity of the solution proposed.

Rich Internet Applications behave as black boxes of data, which makes the access and processing of these data difficult (data interoperability), especially to some types of clients on the Web (such as search engines or accessible readers). The only method to gain access (i.e., crawl and index) to all the textual content of these event-driven applications is through a visual representation of the data produced by the Web browser, which cannot be employed by software systems (e.g., an application can contain animations shown in the user interface depending on the user data). Some authors[2] have provided solutions to this problem by partially building a graph-based representation of the navigational structure of the application. Such representations have a high cost of computation and are still incomplete (e.g., they cannot access the data hidden behind a search form).

The enterprises behind the development of the Web searches and the RIA technologies (mainly Adobe with Flex) have also worked on solving this issue. However, the results of these efforts are not clear. They do not clarify what type of content can be accessed and the manner in which these applications should be developed.

The same problem arises in multimedia searches in RIA. RIAs extensively employ multimedia elements, i.e., images, videos or music, which complement the textual information they present. RIA technologies facilitate the use of these elements that help users to assimilate the information transmitted by the designers. Nonetheless, given their intrinsic structure, the data exposed through these elements remain unreachable to search engines.

In this scenario, the first objectives are to improve the exportability of the textual data visualised in a RIA and the information describing the multimedia elements it contains, in such a way that external software agents (Web search engines and accessible readers) could use it.

These objectives could be achieved in a technology-independent manner by reusing technologies, techniques and resources from another Web trend, i.e., the Semantic Web and, more specifically, the Web of Data. During this last decade, new techniques and technologies for knowledge capturing, storage, management and interoperability have been developed, which can be reused in RIA. For instance, the use of ontologies and the techniques for ontology development reached a sufficient level of maturity to be reused in the first two objectives. In the case of multimedia elements, there are already some approaches that deal with the problem of sharing information about multimedia elements on the Semantic Web.

[2] An analysis of the different approaches was performed by Choudhary et al. (Choudhary et al., 2012)

The solution proposed should be validated adequately in order to ensure its feasibility, detect the possible downsides and propose improvements that avoid them if possible. In this case, the chosen assessment method is based on the development of a collection of case studies (e.g., the development of a media player or a social network site; for a complete description please check Section 3.4, page 53), because it can simulate the functionalities of RIA in real scenarios.

Objective *2) Design a model-driven methodology for the development of the solution.*

O2.1) Facilitate the development of the solution proposed in O1.

O2.2) Improve the maintainability of the solution proposed in O1.

O2.3) Extend an existing methodology for the development of RIA.

By addressing the development of the proposed solution from the perspective of the Web engineering, the associated costs and risks can be reduced, thus facilitating its adoption. Model-driven methodologies have proven to be effective for the development of complex Web applications such as Rich Internet Applications or Semantic Web applications. Several methodologies dealt with the design and development of the different concerns: from data persistence to interface design and personalisation (for a detailed description, please, see Section 2.2).

In real scenarios, the features of Web applications can be updated or modified according to the changing requirements of the stakeholders. The need to combine the requirements of two Web trends, which manage techniques of data visualisation with techniques for knowledge representation, increases the complexity of developing and maintaining the solution proposed in order to achieve O1. The choice of a model-driven methodology can facilitate its development in these scenarios of frequent changes of requirements. Model-driven methodologies are capable of dealing with all the desired features of the application by means of software models and, in this way, any modification to the application will only involve changes to the underlying models.

Given the known benefits and the efforts already spent in the field of Web engineering, another objective is to reuse and extend one of the existing methodologies for the development of Rich Internet

Applications. In this way, the effort involved in designing a new development process and its artefacts can be minimised.

Objective *3) Develop a CASE tool capable of supporting the methodology designed.*

Implementing the methodology proposed in O2 in a software tool is necessary in order to spread its adoption. Model-driven methodologies define a collection of artefacts, i.e., mainly models and transformations, by means of which designers can generate complete applications or specific software components. For this aim, the methodology should be accompanied by a CASE tool that supports and automates the creation and editing of the models as well as the processes of model transformation and code generation. The challenge behind this objective is to develop a tool that could effectively combine the design of components with techniques of knowledge representation (including reuse of external knowledge) and rich user interface design.

The process of development of the CASE tool can be used to validate the methodological proposal and detect those aspects that could be improved from the perspective of the designers. Furthermore, the software tool could be easily shared with other researchers, thus facilitating the assessment of the methodology by other users or its application to business scenarios.

Most of the existing model-driven methodologies have been implemented in a CASE tool. Following the same reasoning explained in O2, reusing an existing implementation would facilitate and speed up the achievement of this objective.

1.2 The Structure of the Dissertation

The contents of the present dissertation are organised in seven chapters and eight annexes, which describe the solutions proposed to the problems detected in order to achieve the objectives of this thesis. The following chapters of this manuscript are structured as follows:

- *Chapter 2: State of the Art.*
  This chapter analyses the state of the art of the different topics addressed in this thesis. It is divided in two main sections: the first one deals with Web applications, and the second one describes model-driven methodologies for engineering Web applications.
  In the first section, the chapter explores the two different types of Web applications involved in this thesis, starting from Rich Internet Applications, their main features and the problem of searching their contents from the perspective of several authors and their approaches. Subsequently, the same analysis is repeated, this time focusing on the Semantic Web applications and those aspects of the Semantic Web (including the Web of Data) that could be used to propose a solution. Finally, this section describes a scenario in which the features of both types of applications are required, i.e., Web applications for Business Intelligence.
  The second section introduces the main model-driven methodologies employed to develop of each of these application types, and describes their development process and main models. It also specifies whether they are supported by a CASE tool.

- *Chapter 3: Rich Internet Applications on the Semantic Web*
  The first contribution of the thesis is introduced in this chapter: a new type of Rich Internet Application called *Semantic Rich Internet Applications (SRIA)*, which employs Semantic Web technologies to solve the issues identified in the first two chapters. This chapter focuses on describing the requirements of this type of application, its structure (its main components) and the use cases employed for the assessment of the proposal.

- *Chapter 4: A methodology for the development of Rich Internet Applications on the Semantic Web.*
  Chapter 4 presents the S^{m}4RIA methodology, i.e., an extension of the OOH4RIA model-driven methodology addressing the development of SRIAs. This chapter describes the development

process proposed, the S^{m}4RIA metamodel and the main models involved in the process. Moreover, the chapter introduces two configurations of S^{m}4RIA: one for the development of Business Intelligence applications; and another oriented towards the modernisation and reengineering processes.
The following three chapters present each of the activities of the S^{m}4RIA development process, detailing all the aspects involved over the same case study: the development of a social network site.

- *Chapter 5: Designing the server components of a Semantic Rich Internet Application.*
  This chapter describes the first activity of the S^{m}4RIA methodology, which is focused on the design of the SRIA server by means of three models: the Domain model, the Extended Domain Model and the Extended Navigational Model. The sections of this chapter explain the creation of the models and the elements of their metamodels, presenting the abstract and concrete syntaxes and using the development of a social network site as a case study.

- *Chapter 6: Designing the client components of a Semantic Rich Internet Application.*
  This chapter describes the second activity of the S^{m}4RIA methodology, which is focused on the design of the SRIA client, including the user interface, by means of two models: the Extended Presentation Model and the Extended Orchestration Model. The sections of this chapter explain the creation of the models, describing the abstract and concrete syntaxes, by continuing the development of the social network site described Chapter 3.

- *Chapter 7: Generating the software modules of a Semantic RIA: transformations and implementation.*
  This chapter describes the third activity of the S^{m}4RIA methodology, which is focused on the generation of the SRIA software modules from the models designed in chapters 5 and 6.

This chapter is divided in three main sections. The first one explains the software architecture of a typical SRIA, based on the experience gained in the development of the case studies. It describes all the modules and architectural patterns applied. Using this architecture as a reference, the second and third sections describe the transformation processes included in S^{m}4RIA, both model-to-text and model-to-model, respectively.
This chapter offers an overview of all the transformation processes included in the methodology and includes the concrete definition for some transformation rules.

- *Chapter 8: S^{m}4RIA extension for OIDE.*
  Chapter 8 introduces the CASE tool "*S^{m}4RIA extension for OIDE*", which implements the model editors and the transformation processes of the S^{m}4RIA methodology using the Eclipse framework as a basis. This chapter illustrates the main features of the tool using a series of screenshots.

- *Chapter 9: Conclusions and Future Work.*
  This chapter summarises the main contributions of this thesis by answering each of the research questions and analyses the degree of achievement of the established objectives. Finally, in the light of the conclusions drawn, the main lines of future work are described.

Furthermore, the manuscript contains a set of annexes that extend and clarify the information included in the chapters:

- *Annex A. Scientific Contributions*
  The first annex lists the scientific publications resulting from the research activities derived from this thesis.

- *Annex B. Main elements of the Navigational & Visualisation Ontologies.*
  This annex describes the Navigational and Visualisation ontology, introduced in Chapter 3.

- *Annex C. Description of the SRIA Case Studies.*

Annex C describes some of the case studies used in the evaluation of the SRIA proposal. The case studies are previously introduced in Chapter 3.

- *Annex D. Design Models Resulting From the Case Studies.*
  This annex contains the S^{m}4RIA models for the development of some SRIA case studies.

- *Annex E. The Extended Presentation Metamodel: Abstract Syntax.*
  Annex E contains the schemas of the complete abstract syntax of the Extended Presentation Metamodel.

- *Annex F. Transformation Rules.*
  This annex includes the code of the model-to-text and model-to-model transformation rules that was not introduced in Chapter 7.

- *Annex G. Implementation Details.*
  Annex G contains some details of the implementation that were not introduced in chapters 7 or 8 as well as a brief reference of the main elements of the languages for the definition of transformation rules.

- *Annex H. Resumen en español.*
  This last annex includes an extended abstract of the manuscript in Spanish.

Chapter 2. STATE OF THE ART

This chapter contextualises the problems addressed by this thesis and the solutions proposed by different authors. This analysis motivates and constrains the decisions taken in the following chapters. All the information analysed will be the basis for the proposal of a solution to the issues identified in the first chapter.

The chapter begins with the analysis of the two new types of Web applications that involved a technological evolution in the last decade. First, the analysis explores the concept of Rich Internet Application, their features and open issues. Subsequently, the chapter aims at describing the concept of Semantic Web Application, whose features can help to solve the issues found in RIA, and a specific scenario that requires the features of both types of application, i.e., Web applications for business intelligence.

The second part of the chapter addresses the description of the methods of development of the types of application analysed in the first part: RIA and Semantic Web applications. Specifically, the chapter introduces the main model-driven methodologies for each of the application types, as well as their main objectives, activities and models.

2.1 ON WEB APPLICATIONS

During the last decade, the Web 2.0 (O'Reilly, 2005) considerably extended the use of the Web among a large variety of users. Within this broad concept, at least two sub-trends could be identified, one associated to a change in the users' behaviour and another technological:

the Social Web and the Rich Internet Applications (RIA). Rich Internet Applications presented several benefits associated to their user interfaces, but also reintroduced some issues of exportability of contents already solved in traditional Web applications. The development of the Semantic Web created new ways of sharing knowledge across the Web, which have not been fully considered for the development of RIA.

This section firstly describes the concept of RIA and its main features and problems on the current Web scenario in a generic manner. Subsequently, it explores the concept of Web applications on the Semantic Web emphasising those features that can help to solve RIA issues. Finally, the section analyses the use of RIA in a specific scenario, i.e., in the applications for business intelligence, in which the issues detected are of special interest.

2.1.1 Rich Internet Applications

With the evolution of the Web architecture and particularly the user interfaces, a new type of application called Rich Internet Application appeared. The concept of rich client (and rich internet application) was firstly introduced by Allaire (Allaire, 2002), whose main features were the ability to use external Web services, the possibility to use connected and disconnected clients (i.e., clients that contain the software modules required to perform some tasks independently from the server) and the use of complex graphical representations. At that moment, those features were bound to a single technology, i.e., *Macromedia Flash MX*, also introduced by Allaire.

From that initial approach, the concept has grown and evolved together with the technologies and frameworks for its development. New technologies appeared, bringing new improvements in the user interfaces, while others fell in disuse. Due to their increasing size and requirements, their development was also addressed from the engineering view point and several methodologies of development were designed. The concept of Rich Internet Application became widely used and each approach personalised the definition on the same initial basis. Busch and Koch in their survey (Busch and Koch, 2009) presented a set of the main definitions found in scientific articles, pinpointing the positive aspects and the ambiguities of each. From these, they proposed their own unifying definition.

After a decade of wide use, Rich Internet Applications can be defined as Web applications with the following features:

1. The application data can be processed by either client or server components.
2. The communication processes between server and client modules follow asynchronous protocols, in such a way that the interface is not blocked while waiting for the server data.
3. Their user interfaces have a *look-and-feel* (aesthetic features) similar the one of desktop interfaces, using a wide set of interactive presentation elements and embedding multimedia elements, as well.

This definition is supported by different research works (Fraternali et al., 2010a; Hermida et al., 2011a; Meliá et al., 2008) and is a simplification of Busch and Koch's definition focusing on the main features of the RIA. The features mentioned in this definition are desirable for any RIA. As explained by Toffetti et al. (Toffetti et al., 2011), depending on the degree of fulfilment of these requirements, RIA can be classified into different types:

- *Traditional Web applications with RIA-makeover*: applications that use RIA capabilities for some operations (usually asynchronous communication processes, e.g., Facebook) or embed RIA snippets (e.g., Flex advertisements).
- *Rich UIs*: Web applications with widget based UIs where the client-side logic is an extension of the browser that supersedes its core responsibilities such as managing states and handling events (e.g., Gmail).
- *Standalone RIAs*: Web applications capable of running inside and/or outside the browser connected or unconnected to the server.
- *Distributed RIAs*: these are applications whose data and logic are distributed across client and server (sometimes dynamically). They enable online collaboration among users (e.g., Google Wave).

From the original contribution of Macromedia Flash, several technologies for the development of this type of applications emerged and have been improved after several versions, including Flash/Flex. The analysis of the RIA applications of Busch and Koch (Busch and Koch, 2009) and Toffetti et al. (Toffetti et al., 2011) introduce an overview of the current technologies. Each of these technologies provides a set of features and visual elements for the definition of rich user interfaces and rich clients, which have several similarities. Although they can invoke Web services from different types, the client technologies are incompatible among them. According to the implementation technologies, RIAs can be classified into three types (Busch and Koch, 2009; Hermida et al., 2011a; Toffetti et al., 2011):

- *Browser-oriented RIAs*: This type of RIAs are visualised in Web browsers using the basic built-in components included. They are based on HTML and JavaScript technologies (i.e., AJAX) and frameworks (e.g., jQuery[3]). Their basic components and software modules are stored as text files that are directly interpreted by the browser.
- *Plug-in-oriented RIAs*: This type of RIAs are visualised in the Web browser but, this time, users need to install a special extension, normally called plug-ins, which actually renders the information of the application in the browser. Their basic components are binary objects that can only processed by its corresponding interpret. Each technology (e.g., Adobe Flex[4], Microsoft Silverlight[5] or OpenLaszlo[6]) generates its own binary objects and plug-ins which are incompatible among them.
- *Desktop RIAs*: This type of RIAs can visualised with the Web browser when the user is on-line, or downloaded and executed off-line using a special framework. They could be considered as a special type of plug-in oriented RIA, since their structure, as well as the technologies in which they are implemented, e.g., Adobe Air applications[7], are similar.

[3] jQuery Web site: `http://jquery.com`
[4] Adobe Flex Web site: `http://www.adobe.com/en/products.flex.html`
[5] Microsoft Silverlight Web site: `http://www.microsoft.com/silverlight/`
[6] OpenLaszlo Web site: `http://www.openlaszlo.org`
[7] Adobe Air Web site: `http://www.adobe.com/en/products/air.html`

During the second half of the 90's, current Web searchers replaced the old Web directories and gradually became the main gate for users to access the information stored on the Web. When users search for information, they mainly trust in the services provided by Web search engines (or Web information retrieval systems), which, from a collection of user keywords that express the main informational needs, they can retrieve a list of the Web documents which probably contain the information searched. These issues related to the searchability of the contents of RIAs can prevent developers from implementing complex RIA applications and enterprises from adopting them for showing their products and services.

Despite the benefits of the RIA features mentioned in this section, these applications have not been widely adopted by the developers. Traditional Web applications with RIA-makeover are a very frequent type of RIA, which offer an acceptable level of interactivity to users. JavaScript-based RIA technologies are commonly used (now with HTML5) and Adobe Flex and Microsoft Silverlight objects are also numerous. However, the adoption of RIA has been lower than expected due to the problems detected with the main Web search engines such as Google or Bing, which discourage the use of the main RIA technologies (AJAX, Flex or Silverlight) in the Web applications[8] [9] [10]. Web search engines crawl and index the text found in the Web sites, mainly embedded in HTML code, and they have certain limitations when crawling the content of RIAs. Crawling in RIA cannot be performed following the URLs contained in a Web page, since they have an internal state which cannot be explored following hyperlinks in the HTML code (RIAs can keep the same address to visualise different information). The information shown in RIA UIs is driven by the user events (mainly raised by mouse or keyboard interactions), which complicates the access to the data independently from the chosen technology. Furthermore, in the case of plug-in-oriented RIAs, the information visualised is stored in the binary objects of the applications which can be only interpreted by a browser extension. Granting access to the RIA server service could be a

[8] Flash and other rich media files – Webmaster tools:
`http://support.google.com/webmasters/bin/answer.py?hl=en&answer=72746`
[9] AJAX – Webmaster tools:
`http://support.google.com/webmasters/bin/answer.py?hl=en&answer=81766`
[10] Bing indexing advice – General questions:
`http://www.bing.com/community/webmaster/f/12248/t/658480.aspx`

solution but it might compromise the data security. Sometimes, it is also possible to include metadata in the HTML code (normally in the HTML header) that contains the RIA object with the main keywords related to the application, which offer a biased view to the search engines.Other software components might need to use the text of the application for performing their tasks and, in the same way, they will not be capable of accessing such information; e.g., the automatic Web readers for people with physical impairments, which facilitate the access to the information to people with different problems.

The difficulties of exploring the information shown in RIAs (i.e., crawling the content of a RIA) have already been studied by several authors addressing different tasks over AJAX RIAs. For instance, Mesbah et al. (Mesbah et al., 2012, 2008) are focused on the processes of crawling over AJAX applications in order to improve Web information retrieval. As another example, Benjamin et al. (Benjamin, 2010; Benjamin et al., 2011) oriented their work on this task towards improving the processes of automatic testing of RIA. A further analysis on this topic was performed by Choudhary et al. (Choudhary et al., 2012), which studied the state of the art on this task and the challenges that still need to be addressed.

The new methods analyse the structure of the Web site and crawl across the content being aware of the changes of the content produced by the user events. Despite the efforts spent on this task, current approaches cannot provide access to all the information contained in a RIA in an efficient manner (in terms of time and computational resources) and can only be applied to browser-oriented RIAs, whose basic components are similar to the ones of traditional Web applications with a textual representation (HTML code). Still, plug-in-oriented RIAs have not been addressed by the research community, given that the technologies for building this type of RIAs are generally proprietary. In 2008, Google and Adobe, owner of the Flex technology, announced the improvement of the indexing of the text contained in Flex RIAs[11] [12]. However, the extent of this improvement is not clear and whether

[11] Official Google Webmaster Central Blog: Improved Flash Indexing.
`http://googlewebmastercentral.blogspot.es/2008/06/improved-flash-indexing.html`

[12] Adobe Advances Rich Media Search on the Web. Press release.
`http://www.adobe.com/aboutadobe/pressroom/pressreleases/200806/070108AdobeRichMediaSearch.html`

Google is actually using it, since they still recommend not using this type of applications[13].

The rise of the HTML5 standard, which extends and improves the current HTML standard, with the aim of simplifying the creation of RIAs, has relegated this problem to a second class in terms of relevance and popularity. Despite the expectation behind HTML5, it cannot solve all the issues of AJAX Web applications and it will probably not extinguish plugin-oriented technologies in the short- or medium-run.

2.1.2 WEB APPLICATIONS ON THE SEMANTIC WEB

After a decade of research and development, the Semantic Web (Berners-Lee et al., 2001)[14] experiences a process of rapid change and evolution looking for a stable state in which the contributions made along these years could be exploited by the industry. The initial vision of the Semantic Web considered software agents as first class Web users and, as such, the information of the Web should be also translated so that could be unambiguously processed and "understood" by them.

Describing the architecture of a typical Semantic Web application is not a simple task, since there is no available standard proposal and each researcher/developer therefore uses the one that best fits with his/her goals. The World Wide Web Consortium joined the initial efforts of standardisation and proposed a seven-layer architecture of the Semantic Web, introduced by Berners-Lee (Berners-Lee, 2000), as a route map for researchers and developers. This layered architecture mainly expresses the desirable features of the whole Semantic Web as a mixture of different technologies and features, which comprised from low-level details, such as the representation of the characters, to high-level features, such as trustworthiness and security of the sources of knowledge. Two of the pillars of this architecture are the RDF data model for the representation of data objects as subject-predicate-object triplets and the use of ontologies for the representation of the knowledge managed by the applications (see Figure 2.1). Still, this proposal has not

[13] As Google indicates in its Webmaster tools Web site: `http://support.google.com/webmasters/bin/answer.py?hl=en&answer=72746`

[14] Although this citation is the most referenced in the field, the concept of Semantic Web appeared years before, e.g., in a Berners-Lee's report (Berners-Lee, 1998a) and the research carried out in the SHOE project (`http://www.cs.umd.edu/projects/plus/SHOE/index.html`)

been totally standardised even thought it was widely accepted as a starting point. Authors such as Horrocks et al. (Horrocks et al., 2005) or Gerber et al. (Gerber et al., 2008) have proposed some refinements based on the experiences in real projects.

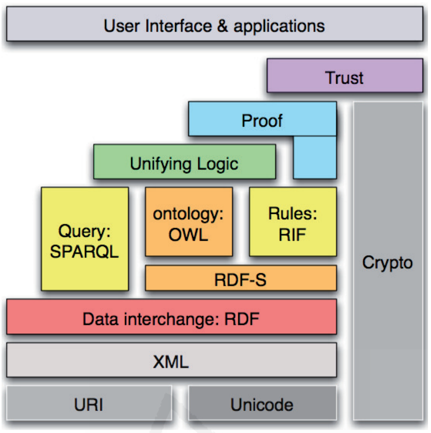


Figure 2.1. Semantic Web architecture in 2006 by Tim Berners-Lee[15].

Traditional Web applications should have been adapted to the W3C architecture, thus including new annotations in their code that link the concepts explained in the text to the elements of the ontologies available on the Web. However, there is no consensus on the concept of semantic (or Semantic) Web application in the Web community. At present, it is possible to find different viewpoints of the same concept. Some authors, such as d'Aquin et al. (D' Aquin et al., 2008) and Kozaki et al. (Kozaki et al., 2008), describe them in a general manner as applications of the Semantic Web, which refer to those applications that use the available Semantic Web technologies and resources (ontologies, knowledge bases, etc.) Other authors (Brambilla and Facca, 2007; Corcho et al., 2006; Frasincar et al., 2010; Lausen et al., 2005) show a common viewpoint, more related to the field of Web engineering. For them, a semantic Web application is a Web application that implements the Semantic Web architecture, completely or partially, and employs technologies associated to the Semantic Web architecture. From these previous definitions, in this thesis, a Semantic Web application is defined as a

[15] From the Tim Berners-Lee's presentation at AAAI 2006: `http://www.w3.org/2006/Talks/0718-aaai-tbl/Overview.html#(14)`

Web application[16] of the Semantic Web which is capable of sharing its contents as ontology-based annotations to Semantic Web agents.

Other research approaches studied the possible features and the structure/architecture of the Semantic Web applications as a first step towards the standardisation of this type of applications with the aim of improving their development by means of software engineering techniques (or more specifically, techniques of Web Engineering). Brambilla and Facca (Brambilla and Facca, 2007) defined a set of requirements and features for developing Semantic Web portals, which facilitate the development using a model-driven approach. Heitmann et al. (Heitmann et al., 2009) analysed the different software modules contained in current Semantic Web applications and subsequently propose an architecture as a basis for the development of future applications. One of the latest studies on this topic was presented by Rovan et al. (Rovan et al., 2011), which proposes a categorisation of the existing types of applications according to their purpose and functionalities. From the Social Semantic Web subfield[17], Kinsella et al. (Kinsella et al., 2009) described and analysed specific types of semantic Web applications developed, such as semantic wikis or semantic blogs.

It is a fact that nowadays, one decade after the publication of the view of the Semantic Web, the Semantic Web architecture is still uncompleted, upmost layers of the architecture, which are the most complex, have not been designed or standardised and the adoption of the lower ones is slowly increasing. By the end of the 2000's, a new extension of the Web appeared as a subset of the Semantic Web, which was called the Web of Data[18]. The idea underpinning this initiative is to create a Web not based on documents (i.e., Web pages), but on data, in which it would be possible for users to easily navigate though data of different sources and natures using links. Another aim of the Web of Data is to make public and freely available the huge amounts of data that are currently distributed across different applications and resources using Semantic Web technologies in such a way that could be

[16] Defined by Isakowitz et al. (Isakowitz et al., 1998), equivalent to the concept of Web Information System (WIS).

[17] The Social Semantic Web (Mikroyannidis, 2007) is a combination of the approaches of the Social and the Semantic Web. The community of research on this topic is focused on applying technologies of the Semantic Web to social applications; or using social approaches to continue the construction of the Semantic Web (or the Web of Data).

[18] Heath and Bizer (Heath and Bizer, 2011) offer a good introduction of the concept of Web of Data.

automatically shared and reused. This will enable users and applications to obtain new, valuable knowledge from the reuse of the data (and knowledge) stored on the Web.

The Web of Data presents a more realistic and short-timed vision of the Web that can be reached by using the technologies already developed for the Semantic Web architecture. The architecture of the Web of Data reuses the four lowest layers of the original Semantic Web architecture, which contain the main techniques and technologies for sharing knowledge on the Web (see Figure 2.1: from URI and Unicode to SPARQL, OWL and RIF). This facilitates that the applications, which can be considered Semantic Web applications as well, share a common structure independently their goals.

Apart from the use of Semantic Web technologies, the Web of Data is founded over the concept of Linked Data, which refers to a collection of good practices for publishing and linking data structures on the Web proposed by Berners-Lee (Berners-Lee, 2006), which have been developed by the research community by the end of last decade. The four main principles are the following:

1. Use URIs as names for things.
2. Use HTTP URIs, so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs, so that they can discover more things.

Based on these four principles, data from different nature can be published on the Web as linked datasets. Apart from the Berners-Lee's Web site, there are two main information sources about Linked Data and the existing datasets on the Web of data: the *linkeddata.org* Web site[19] and the *LinkingOpenData W3C Community Project*[20], which coordinate the efforts of the community and the diffusion of the Linked Data concept, as well as, a register with the main datasets[21] of the Web of Data.

[19] `http://linkeddata.org`

[20] `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

[21] An overview of the datasets available on the Web of Data can be found on the following web site: `http://richard.cyganiak.de/2007/10/lod/`

During the past five years, the Web of Data has experienced a constant growth in the number of datasets freely available[22], supported and boosted by different organisations (with public or private funding such as the UK Government) and users (from academia or industry). The existing dataset network is continuously enriched with data of any nature and topic from local databases, aiming at improving the permeability of the data in our societies. At present, there are several data repositories storing information on a wide variety of topics, e.g., from statistics published by their national governments (e.g., data.gov from the US government; and data.gov.uk, from the UK government) to the number of music records of their favourite artist (e.g., MusicBrainz or Jamendo).

To this end, several approaches and applications have already dealt with processes of transformation that turn our closed databases into linked datasets on the Web (Barrasa Rodríguez, 2007; Berners-Lee, 1998b; Bizer and Seaborne, 2004; Būmans and Cerāns, 2010; Volz et al., 2004; Xu et al., 2006).

Still, a large amount of information flowing through the Web is managed by traditional Web applications. In order to adapt them to the Web of Data (these applications could be considered as legacy Web applications), developers could directly extend the features of the Web applications or simply employ one of the existing tools (e.g., OpenLink Virtuoso Server[23], D2RQ[24] or ODEMapster[25]), which are capable of translating the tuples of a relational database into ontology instances given a set of mapping rules.

The development of rich clients (clients with rich interfaces) in applications of the Semantic Web or the Web of Data is primarily oriented to the visualisation of data aggregations or mash-ups (mostly based on JavaScript technologies as can be noticed from an analysis of the last international mash-up competitions[26] [27]). However, the study and application of Semantic Web technologies in Rich Internet

[22] The evolution can be visualised from: `http://richard.cyganiak.de/2007/10/lod/#history`

[23] OpenLink Virtuoso Web site: `http://virtuoso.openlinksw.com/`

[24] D2RQ Web site: `http://d2rq.org`

[25] ODEMapster: `http://mayor2.dia.fi.upm.es/oeg/index.php/es/technologies/9-r2o-odempaster`

[26] AI Mashup Challenge 2012: `https://sites.google.com/site/aimashup12/home`

[27] AI Mashup Challenge 2011: `http://sites.google.com/a/fh-hannover.de/aimashup11/`

Applications has not been taking into consideration in depth. In this area, it is worth mentioning the research carried out by Linaje et al. (Linaje et al., 2009a), who presented the requirements and the changes needed to include semantic (ontology-based) annotations within AJAX user interfaces by applying the W3C WAI-ARIA standard for RIA accessibility (World Wide Web Consortium, 2011).

2.1.3 RIA FOR BUSINESS INTELLIGENCE

Business intelligence (BI) applications have been traditionally focused on the analysis and exploitation of the local business data in order to obtain valuable knowledge about different enterprise areas (marketing, human resources, etc.) that support decision-making processes. To this aim, a wide set of data analysis/mining/visualisation or data warehousing techniques have been developed and applied.

In contrast to these traditional approaches, in the present Information era, enterprises live in an increasingly globalised environment, in which the Web has become the main communication platform, linking information and services among enterprises and their potential clients. Business information systems are progressively ported to the open Web, which improves the availability of the applications and the access to external business data and the new types of information and services offered by other organisations[28].

In this context, BI Web applications need to include a set of specific functionalities that facilitate the management of knowledge from the Web and the publication/access to external Business-to-Business (B2B) and Software-as-a-Service (SaaS) services [29] . In addition, these applications require user interfaces (UI) that support the visualisation (in a user-friendly manner) of different types of data/knowledge.

As shown by Raspal (Raspal, 2010), White (White, 2009) or Laurent (Laurent, 2010), Rich Internet Applications start to play a relevant role as BI Web applications due to their intrinsic features: UIs with high level of

[28] Davenport (Davenport, 2000) presented an overview of the Enterprise systems, as well as his predictions about the future of these systems. Some of the aspects he mentioned are still challenges in this decade, e.g., the use of techniques for knowledge management (Barjis et al., 2011).

[29] The Software & Information Industry Association provided one of the first definitions of this concept and a complete overview (Software & Information Industry Association, 2001).

interactivity and usability (comparing with traditional Web applications) and the asynchronous communication between the server and the client of a Web application, led by a set of mature technologies. RIAs have proven to be an appropriate platform to visualise data from different sources, whose rich user interfaces offer experiences similar to the ones provided by desktop interfaces.

Business scenarios on the Web need solutions with proven effectiveness and specifically adapted to their requirements, helping them to achieve their present goals and improve their future strategy. RIAs are not as specialised in the management of knowledge from the Web as BI systems require. The BI RIAs should deal with issues of knowledge management (KM) and visualisation that finally require complex software modules combined in a single architecture. Thus, the complexity of the architecture notably increases the cost of development and maintenance, which is a risk factor that threatens the viability of this type of applications and their future success.

In the Web field, KM techniques have been developed by the Semantic Web community, which has successfully applied these techniques to the management of the enterprise knowledge and the development of KM applications, as described by Penella et al. (Penela et al., 2011) and Allemang (Allemang, 2010).

KM activities are focused on the organisation of the knowledge contained in the whole enterprise and the creation of new knowledge, in a manner that facilitates the development of business processes. These activities have a direct impact on the employees, who need to carry out a special effort in order to implement and deploy them in the core of the enterprise. The combination of techniques from the Semantic Web with approaches from the Social Web or Web 2.0 – also aligned to the concept of Social Semantic Web, or Web 3.0; (Mikroyannidis, 2007) – helps to implement KM activities on an enterprise, as demonstrated by some existing proposals, e.g., Yammer[30] or MiKrow (Penela et al., 2011).

Yammer is a social network site successfully introduced in several companies that facilitates the communication of the employees as well as the organisation of the information they use in their daily activity. In a similar manner, MiKrow is a micro-blogging tool that improves the

[30] Yammer, the enterprise social network: `https://www.yammer.com/product/`

processes of communication and information sharing between the employees of a company. This tool automatically relates the content to the concepts normally used to carry out their tasks.

Other types of initiatives deal with the management of the enterprise knowledge in order to improve the productivity level of the employees. Some research projects have tackled this problem using KM techniques, e.g., ACTIVE (Simperl et al., 2010) or NEPOMUK (Groza et al., 2007). ACTIVE aims to study methods for introducing KM technologies and applications in the employees' daily tasks in a way that it does not increase the cost of the task, measured in terms of effort or time.

As regards the approaches originating from the Semantic Web field, the continuous growth of the Web of Data has led to new solutions for data sharing and reusing. Allemang (Allemang, 2010) shows the impact of Semantic Web technologies on the business scenario and proposes some adaptation for the publication of enterprise information of an enterprise as linked data. This opens a path towards a scenario in which enterprises and public organisms seamlessly share their data.

2.2 On Model-Driven Web Engineering

One of the aims behind the definition of the requirements and the standard architecture of a Web application is the subsequent standardisation of its development processes, which might spread the use of these applications. The adoption of a new type of applications in business scenarios depends mainly on its maturity, benefits and development costs and risks. The application of model-driven methodologies to the development of Web applications can provide a solid framework for mitigating these risks. Model-driven engineering methodologies facilitate the representation of all the concerns of an application as software models, which can be subsequently employed to automatically generate the code of the application. In this manner, the cost of development and maintenance of the application can be minimised, thus encouraging the use of RIA platforms.

This section describes and analyse the main development methodologies for Rich Internet Applications and Semantic Web applications, including its main features and design process. The description will be divided in two subsections, each of which addresses the development of one type of application.

2.2.1 Model-Driven Engineering of Rich Internet Applications

The development of Rich Internet Applications from the Software Engineering (or Web Engineering) viewpoint is a relatively new research field. The existing model-driven methodologies could be classified in four groups as Busch and Koch (Busch and Koch, 2009) indicate:

a) Specialised extensions to existing methods for developing Web applications.
b) Methods focused on the development of the RIA client and rich user interfaces, which reuse other methods for developing the RIA server.
c) New methods for developing RIAs.
d) Pattern-based approaches.

This section pinpoints the main methodologies proposed offering an overview of its main features.

2.2.1.1 WebML

WebML – *Web Modelling Language* (Ceri et al., 2000) – is a model-driven methodology for the development of data-intensive Web applications. It defines a waterfall software process of seven stages through which developers can conceptualise different aspects of a Web application. This methodology is based on four models (Ceri et al., 2002):

- The Data model, which represents the main data structures of the application using a notation similar to the Entity-Relation model but also compatible with object-oriented representations.
- The Hypertext model and the Content Management model, which specify the organisation of the contents (in terms of Web pages and links) and the business logic of the application.
- Finally, the Advanced Hypertext model, which addresses issues related to the manner of navigation and the effects over the application.

During the last decade, the original WebML methodology has been extended with primitives for modelling different aspects of Web applications or even new types of application (Ceri et al., 2009). Bozzon et al. (Bozzon et al., 2006), Toffetti (Toffetti Carughi, 2007) and Fraternali et al. (Fraternali et al., 2010a) introduce an extension of the method to tackle the development of Rich Internet Applications. Specifically, they proposed an extension of the Data model to facilitate the representation of distributed and temporary data objects, and the Hypertext model, which include new primitives to manage the new types of data objects. Moreover, the authors propose to include a new model for defining the dynamic behaviour of the elements of the RIA user interface, i.e., the RIA dynamic model, similar to UML activity diagrams.

This methodology is supported by a CASE tool called *WebRatio*[31] (Acerbis et al., 2007).

2.2.1.2 UWE

UWE – *UML-based Web Engineering*, proposed by Koch and Kraus (Koch and Kraus, 2003, 2002) is a model-driven development methodology for Web applications based on the use of UML models. More specifically, the methodology uses a UML profile for the specification of the use cases and the design aspects.

The development process in UWE is driven by the design of five models and the transformations between them:

- *Requirements model*, which extends the UML Use Case model with new stereotypes, used for the specification of the application requirements.
- *Content model*, which is a UML class diagram for the representation of the data objects managed by the application.
- *Navigation model*, which extends the UML class model for the representation of the navigation of users within a Web application.
- *Process model*, which extends the UML class model for the representation of the behaviour of the application and the interaction with the users.

[31] WebRatio Web site: `http://www.webratio.com`

- *Presentation model*, which extends the UML class model for the representation of the structure of the user interface using abstract elements, avoiding aesthetic details.

UWE is supported by the ArgoUWE CASE tool (Knapp et al., 2004), which implements the UML profile and the model transformations.

During the last decade, UWE was adapted to Web applications for business scenarios customisable Web applications and Rich Internet Applications, based on the use of the *Business Process Modeling Notation* (BPMN) standard. There are two extensions of UWE for the development of Web applications: one proposed by the original authors, and another presented by other research group.

Koch et al (Koch et al., 2009) proposed a pattern-based approach for the development of Rich Internet Applications with UWE. The authors proposed the definition of extension points in the UWE models for including references to the development of RIA. The patterns are defined as UML state models which are integrated into the UWE models by means of model-to-model transformations or to the final Web application code by means of model-to-text transformations.

The second approach, proposed by Machado et al. (Machado et al., 2009), extended the UML profile of UWE including the elements required to model the specific features of RIA. The resulting methodological extension was called UWE-R. The authors addressed the design of RIA including new elements in the Navigation, Process and Presentation models, which inherit the structure and behaviour of UWE profile elements.

2.2.1.3 RUX-METHOD

RUX-Method – *Rich User eXperience Method* (Linaje et al., 2007; Linaje Trigueros et al., 2007; Preciado et al., 2007) – is a model-driven methodology focused on the development of rich user interfaces, i.e., the user interface and the client modules of a Rich Internet Application. In order to generate the final user interface, this methodology proposes a three-step process driven by the specification of three models:

- the *Abstract Interface*, which represents the abstract elements of the interface, independent from the chose implementation technology;

- the *Concrete Interface*, which represents the structure and the behaviour of the RIA interface in a platform-independent manner. It is based on three presentations: Spatial (structure of the interface and aesthetic features), Temporal (behaviour of the interface) and Interaction (user interactions). It relates the elements of the interface to the services provided by the RIA server.
- the *Final Interface*, which adapts the abstract elements of the two previous interfaces to a specific technology and relates the elements of the interface with the services provided by the RIA server.

RUX-Method does not support the design and development of the RIA server modules since it is focused on the RIA user interface and client modules. For the development of the RIA server, the authors initially recommended to reuse the Data and Hypertext models of WebML (Brambilla et al., 2008). In order to reduce the dependence between RUX and WebML and prove the flexibility of the method, they also proposed a subsequent adaptation for using UWE models (Preciado et al., 2008).

RUX-Method is supported by the RUX-Tool CASE tool (Linaje et al., 2009b).

2.2.1.4 Other Approaches

Other well-known model-driven methodologies for engineering Web applications were also extended to support the development of Rich Internet Applications. This section presents a brief summary of them.

The OOHDM methodology – *Object Oriented Hypermedia Design Method* (Schwabe and Rossi, 1998) – was also extended to deal with the development of Rich Internet Applications (Urbieta et al., 2007). The proposed approach extends the OOHDM Interface model to address the development of the rich user interfaces by means of Abstract Data Views (ADVs), which represent all the structural elements of the rich interface and are organised hierarchically. In order to represent the behaviour of the interface, the authors proposed to use ADV-Charts, which are state machines that allow expressing interface transformations resulting from user interactions.

The OOWS methodology – *Object Oriented Web Solutions* (Valverde Giromé, 2010) – is a model-driven methodology specialised for the development Web (2.0) applications (backend and frontend) based on OO-Method. Valverde and Pastor (Valverde and Pastor, 2008) proposed an extension of the method that addressed the development of Rich Interfaces. In this case, the authors defined a new model, i.e., the Interaction Model, to specify the components of the rich user interface and their dynamic behaviour. They also defined a collection of usual interaction patterns that might occur between users and the RIA interface.

A complete classification of all the existing methodologies and a brief description for each of them can be found in the analysis performed by Toffetti et al. (Toffetti et al., 2011).

2.2.2 Model-Driven Engineering of Semantic Web Applications

In the same manner that Rich Internet Applications, the design (and modelling) of Semantic Web applications has been studied by the main model-driven development methodologies, which proposed their own extensions. Furthermore, new methodologies have been created for this purpose both in the Semantic Web and the Software Engineering field.

The goal of this subsection is to analyse each methodology highlighting their most relevant features. Given the architecture of the Semantic Web (and also the Web of Data), all these methodologies need to model the knowledge managed by the application by means of an ontology (or a vocabulary). Therefore, these methodologies share an activity (or task) for addressing the design of a (domain) ontology that represents the application. The last subsection will review the main approaches for modelling ontologies using software modelling languages, such as UML.

2.2.2.1 WebML Extension

Brambilla and Facca (Brambilla and Facca, 2007) defined the main requirements and features of semantic Web portals and proposed an extension of WebML to address the development of this type of applications. The solution proposed aimed to be generic and thus reusable by other methodologies. The extension adds two new stages to the process: *ontology import*, which addresses the importation of existing ontologies; and *annotation design*, which defines the manner annotations are included within semantic Web applications. Moreover, the authors extended the Hypertext model with new primitives that manage the data obtained from the ontologies imported and its instances.

In a parallel work, Brambilla et al. (Brambilla et al., 2007) established the requirements and defined a set of WebML primitives that allows the exploitation of semantic Web services in Web applications. The approach relies on the Web Service Modeling Ontology[32] and involves the same stages that the previous work. In addition to this, the authors include new primitives in the Hypertext model in order to consume data from external services.

Regarding the development of Social Web applications, Fraternali et al. (Fraternali et al., 2010b) defined and analysed a collection of software patterns used in existing community-based Web applications. Moreover, they proposed a set of examples that demonstrate the manner they can be implemented using WebML and WebRatio.

2.2.2.2 ONTOWEBBER

OntoWebber (Jin et al., 2001) is one of the first methodologies for developing Semantic Web applications, supported by a tool with the same name. The proposed methodology was divided in four activities:

(i) *Integration*, whose aim is to retrieve data from heterogeneous sources on the Web and transform them into a RDF model;

(ii) *Articulation*, in which the semantic inconsistencies of data sources are solved;

(iii) *Composition*, in which the domain ontology is built and the site views are designed according to modelling ontologies are constructed and site views are created on the underlying data as site models; and

(iv) *Generation services*, in which the new Web applications is generated from the site models.

[32] WSMO. Please see the following web site: `http://www.wsmo.org/`

OntoWebber proposes six models that capture and represent the information necessary for generating Semantic Web applications from the authors´ viewpoint:

- *Domain model*, which models the domain ontology of the application (can be created automatically by analysing the domain data and extracting the underlying concepts, properties and relations);
- *Site View model* and *Navigational model*, which specify aspects related to the navigation of the users through the Web site;
- *Content model*, which associates the concepts of the domain ontology to the Site view and Navigational models;
- *Presentation model*, which specifies the structure and the visualisation of the Web pages;
- *Personalisation model*, which the Web site is adapted to the needs of user depending on different parameters; and
- *Site Maintenance model*, which models the behaviour of the Web site when certain data changes happen.

2.2.2.3 RUX-METHOD EXTENSION

As an extension of RUX-Method, Linaje et al. (Linaje et al., 2009a) presented the requirements and the modifications needed to include semantic annotations within AJAX user interfaces by applying the World Wide Web Consortium WAI-ARIA recommendation (World Wide Web Consortium, 2011), which addresses the accessibility of Web user interfaces by means of a set of ontologies. The solution proposed is implemented, and thus supported, by the *EditSAW tool*, which is a CASE tool for the development of accessible Web sites with its own underlying methodology.

In a second approach, Linaje et al. (Linaje et al., 2011) propose a solution applying *ontoRUX*, i.e., an extension of the WAI-ARIA ontology. The authors implemented their approach in *editRUX*, i.e., a software component embedded in RUX-Tool that enables the designers to include semantic annotations within user interfaces.

2.2.2.4 SHDM

SHDM – *Semantic Hypermedia Design Method* (Lima and Schwabe, 2003) – is an ontology-driven design methodology for building Web applications on the Semantic Web. From a collection of three ontologies, i.e., *domain* (describing the data of the application), *navigation* (defining the structure of the application and the navigation paths) and *presentation* (specifying the visualisation of the Web sites), the method generates all the modules of a Web application.

In subsequent works (Fialho and Schwabe, 2007; W3C Model-based User Interfaces Incubator Group, 2009), SHDM was extended with a collection of primitives for the design of RIAs. This extension increased the number of elements of the presentation ontology and extended the functionalities of the Web generator. In this last version, the authors overcame the lack of annotation in their initial Web applications by means of RDFa (World Wide Web Consortium, 2008a).

It is worth mentioning that the original SHDM proposal aims at generating textual annotations within browser-oriented RIAs, based on HTML and JavaScript. This evidently limits the type of technologies that can be used to generate RIAs.

Recently, de Souza and Schwabe (De Souza Bomfim and Schwabe, 2011) updated the methodology in order to generate Web applications that could work with data from the Web of Data, i.e., fulfilling the Linked Data principles. This approach is implemented by the *Synth* platform, which supports the design and development of applications on the Web of Data.

2.2.2.5 WSDM

WSDM – *Web Semantics Design Method* (De Troyer et al., 2007), formerly known as *Web Site Design Method* (De Troyer and Leune, 1998) – is an audience-driven design methodology of Semantic Web applications. This methodology defines an iterative development process that accounts for the users to which is addressed the application from the first stages. More specifically, the design process is divided in five activities: *a)* Mission Statement Specification, *b)* Audience

Modelling, *c)* Conceptual Design, *d)* Implementation Design and *e)* Implementation.

WSDM employs OWL ontologies (World Wide Web Consortium, 2004) to define the domain model of the applications, which allows designers to directly reuse knowledge from other applications directly from the Web. The Semantic Web applications generated by means of WSDM are semantically annotated using XPointer links referencing ontology elements (Casteleyn et al., 2006). This methodology also deals with the generation of semantic annotations for visually impaired users (Plessers et al., 2005).

2.2.2.6 HERA

Hera (Houben et al., 2003) is a methodology for designing Semantic Web Information Systems (SWISs), defined as Web Information Systems[33] (WIS) that use Semantic Web technologies. Specifically, the method uses ontologies for the representation of the domain knowledge of the application, enabling the reuse and allows the reuse of other existing ontologies. Moreover, the authors propose a method for annotating the Web sites generated.

From the Hera perspective, the architecture of a SWIS can be divided in three layers:

- *Semantic Layer*, which implements the processes of data gathering and integration from different types of sources;
- *Application Layer*, which defines the structure of the application and includes the definition of the adaptation processes; and
- *Presentation Layer*, which includes the modules that generate the presentation of the application for a specific presentation platform, e.g., HTML or WML (Wireless Markup Language).

There are two main activities in the process defined by the Hera methodology: *Data Collection* (Vdovjak et al., 2003), in which data from different sources can be integrated; and *Presentation Generation* (Frasincar et al., 2010), in which the designer builds a hypermedia presentation for the data retrieved in the first phase.

[33] Defined, in this case, as information systems that use the Web paradigm (and technologies) to retrieve information from different sources and deliver it to Web users.

Van der Suijs et al. (Van der Sluijs et al., 2006) proposed an extension of the methodology, called *Hera-S*, *"to support the design of navigation-oriented Web structures over Semantic Web data."* Hera-S enables the generation of Semantic Web applications that, using the Sesame RDF framework (Broekstra et al., 2002) for the management and storage of RDF triples, allow other applications to directly access and modify the data using the SeRQL language – *Sesame RDF Query Language* (Broekstra and Kampman, 2003).

2.2.2.7 ON MODELLING ONTOLOGIES

In the development of Semantic Web applications or Knowledge Management applications, the problem of modelling ontologies is a traversal task, since ontologies capture the semantics of the data structures managed by applications. The differences between modelling and designing ontologies have been discussed widely since they offer mechanisms for knowledge representation.

Cranefield and Purvis (Cranefield and Purvis, 1999) introduced one of the first approaches for the modelling of ontology using existing languages, i.e., UML. The authors suggest the manner in which the elements of the UML class diagram can be used to design ontologies and the limitations of the approach.

In the field of model-driven Software Engineering, Djurić et al. (Djuric et al., 2004) introduced a UML profile for the design of ontologies. This approach was adapted to the Model-Driven Architecture standard in subsequent proposals presented by Gašević et al. (Gasevic et al., 2007, 2005). Here, the authors introduced a process and a domain specific language for the design and generation of OWL ontologies. These pieces of work led to the Object Management Group standard for the definition of a metamodel for designing ontologies called the Ontology Definition Metamodel (Object Modeling Group, 2009).

The use of models for the representation of ontologies opens up some philosophical issues about the differences between meta-modelling and ontology design, which are not clearly addressed in the previous approaches. The similarity of the concepts of model and ontology as well as the similarity of their components may cause some confusion in researchers and developers. To clarify this issue, Kappel et al. (Kappel et al., 2006) and Silva Parreiras et al. (Silva Parreiras et al., 2007) analysed the characteristics of both methods for knowledge representation and proposed a transformation between the MDA[34] framework (UML and EMOF/Ecore metamodels) and the OWL representation language, which is the standard in the W3C architecture for the Semantic Web.

Although these approaches have not been directly applied to the development of Semantic Web applications, they established the basis for modelling ontologies from a generic viewpoint (not applied to a specific task).

2.2.3 Analysis of the Methodologies

Each of the methodologies described in Section 2.2 addresses the problems related to the development of Rich Internet Applications and Semantic Web applications. This section synthesises those aspects of the methodologies more relevant according to the objectives of this thesis.

Table 2.1 and Table 2.2 show the results of this analysis. The first table analyses the features related with the development of RIA, while the second one contains those related to the development of Semantic Web applications. All the features are aligned to the objectives in this thesis, which offers an overview of how they are covered by the current methodologies. As can be appreciated from these tables (and described in this chapter), the existing methodologies contain the features required for modelling rich user interfaces (each using its own approach) and also the features required for developing Semantic Web applications. However, none of them combines the techniques for modelling an application with both types of functionalities (see Table 2.2, "Design rich user interfaces"). Moreover, the applications they generate do not share their knowledge to be consumed on the Web of Data.

The methodology proposed for the achievement of O2 needs to combine these two modelling aspects in an effective manner[35].

[34] Model-Driven Architecture: `http://www.omg.org/mda/`

[35] The final comparison including the methodology proposed in this thesis can be found in Chapter 9, Table 9.1 (page 223) and Table 9.2 (page 224).

Table 2.1. Summary of the methodologies for developing Rich Internet Applications.

	Objective	Feature	Methodologies									
			WebML	**UWE**	**UWE-R**	**RUX-Method**	**OOHDM**	**OOWS**	**SHDM**	**WSDM**	**OntoWebber**	**Hera/ Hera-S**
General		Type	Model-driven	Model-driven	Model-driven	Model-driven	Model-driven	Model-driven	Ontology-driven	Model-driven	Model-driven	Model-driven
	O3	CASE tool	WebRatio	ArgoUWE		RUX Tool					OntoWebber	
Rich Internet Applications	O2	Develop Rich Internet Applications	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
		Type of methodology (Busch & Koch 2009)	a	d	a	b	a	a, extends OO-Method	a			
	O1	Design RIA server	Yes	Yes	Yes	No	Yes	Yes	Yes			
	O1	Design the structure of Rich User Interfaces	Yes, with the Hypermedia model	Yes, with a new set of patterns over the Presentation model	Yes, extending the Presentation model	Yes, with the Concrete Interface	Yes, with Abstract Data Views	Yes, with the Interaction model	Yes, with the Presentation ontology			
	O1	Design the behaviour of Rich User Interfaces	Yes, with the Hypermedia model	Yes, with a new set of patterns over the Navigation and the Process	Yes, extending the Navigation and Process models	Yes, with the Concrete Interface	Yes, with ADV-charts	Yes, with the Interaction model	Yes, with the Presentation ontology			
	O1	Generate browser-oriented rich clients	Yes	Yes	Yes	Yes	Yes	Yes	Yes			
	O1	Generate plugin-oriented rich clients	Yes	No	No	No	No	No	No			
	O3	CASE tool support										

Table 2.2. Summary of the methodologies for developing Semantic Web Applications.

	Objective	Feature	Methodologies									
			WebML	UWE	UWE-R	RUX-Method	OOHDM	OOWS	SHDM	WSDM	OntoWebber	Hera/ Hera-S
Semantic Web Applications	O2	Develop Semantic Web Applications	Yes	No	No	Yes	No	No	Yes	Yes	Yes	Yes
	O1.1, O1.2, O1.3	Design ontologies	Yes, using the Ontology model			No			Yes, with the Domain ontology	Yes	Yes, with the Domain model	Yes, in the Data collection activity
	O1.3	Import ontologies	Yes			No, only works with OntoRUX.			Yes, with the domain ontology	Yes	Yes	Yes
	O1.3	Reuse external knowledge bases	Yes, with Semantic Web Services			No			Yes, in another approach *(De Souza Bomfim & Schwabe, 2011)*	No	No	
	O1.1, O1.2	Generate ontology instances	Yes			Yes			Yes	Yes	Yes	Yes
	O1.1, O1.2	Generate semantically annotated UIs	Yes			Yes			Yes	Yes	Yes	Yes
	O1.1	Generate a service for accessing the ontology instances	Yes, Semantic Web Services			No			No	No		Yes, based on SeRQL
	O1.1, O1.2, O1.3	Aligned to the Linked Data principles	No			No			Yes	No	No	No
	O1.1	Generate Linked Data datasets	No			No			No	No	No	No
	O1.3	Reuse Linked Data datasets	No			No			Yes, in another approach *(De Souza Bomfim & Schwabe, 2011)*		No	No
	O1	Design rich UIs	No			No			Yes	No	No	No
	O3	CASE tool support	-			editRUX, included in RUXTool			Synth platform		OntoWebber	

2.3 CONCLUSIONS

This chapter reviewed the concepts needed to contextualise the issues found in the first chapter and to propose a solution to them. Furthermore, it described the main model-driven methodologies that address the development of the features required to develop the solution designed.

The first part of the chapter started with the description of the concept of Rich Internet Application and the manner it was extended and refined in the last decade reaching the level of maturity sufficient to be address from an engineering viewpoint. As mentioned during the chapter, their user interfaces introduced a set of benefits regarding traditional Web applications that attracted several users and developers (e.g., better UI interactivity, more complex widgets and the use of asynchronous communication processes between clients and servers.) However, due to their intrinsic structure, they reintroduced the problem of accessing the information visualised and retrieve it. Despite the efforts spent in order to solve this issue, from the information analysed, it can be concluded that, at present, there is no solution that proved their validity for any type of RIA. The approaches analysed offer only partial solutions to the content of the RIA and are computationally expensive.

Subsequently, the analysis of the Semantic Web applications illustrated how these applications were designed to share knowledge on the Web based on the architecture of the Semantic Web. The techniques and technologies used in this field could be reused to extend the concept of RIA and, thus, solve the data exportability issues detected and, at the same time, provide an unambiguous representation of the knowledge contained in the application, which could improve the searches of data. This possible approach could mainly solve the problems related to Web searches and adaptability but would not affect other issues such as application testing, which was also a problem in the context of research in RIA. As could be seen, the use of the features of both application types is especially relevant for the development of Web applications for Business Intelligence.

The second part of the chapter was focused on the analysis of the existing model-driven methodologies for engineering Web applications. From this analysis, it could be noticed that none of the methodologies combine the features needed to resolve the problems found in RIA. The studied methodologies address the features required independenly, i.e., without combining the features for the development of RIA and the ones for Semantic Web applications. In most of them, the techniques and technologies from the Semantic Web were considered some years ago and were not updated.

The methodologies that consider most of the requirements are WebML and SHDM. However, WebML does not combine the features for modelling Rich Internet Applications and Semantic Web applications. Moreover, it does not consider the new manners of consuming and producing knowledge on the Semantic Web proposed in the Web of Data.

SHDM shares the same issue with WebML. They propose different solutions to design Semantic Web applications, rich user interfaces and applications that consume linked data from the Web of Data. However, the authors have not proposed a combined solution that could solve the issues found in RIA. With the last extension of SHDM (De Souza Bomfim and Schwabe, 2011), it is possible to create applications that consume data from the Web of Data, but this extension does not deal with the problems related to RIA.

RUX-Method uses the ontoRUX ontology, which extends the WAI ARIA ontology, for annotating rich user interfaces in browser-oriented RIA. Although, this approach uses all the features required, it is oriented to a single goal, i.e., improving the accessibility of the interfaces. RUX-Method generates applications that share information of accessibility in a single manner, and it does not consider some tasks shared by the methodologies for the development of Semantic Web applications, e.g., the design of domain ontologies.

One aspect noticed during the analysis is that none of the methodologies, apart from RUX-Method, designs and generates plugin-oriented RIA. This might be a sign of the problems detected in this type of RIA, whose data is not represented using a textual representation. RUX-Method (with RUX tool) generates Flex applications but do not address the issues of data access related to them.

Chapter 3. RICH INTERNET APPLICATIONS ON THE SEMANTIC WEB

The last chapter analysed the current approaches that could be used to solve the issues detected in RIA. As could be seen, several authors (Fraternali et al., 2010a; Linaje et al., 2007; Meliá et al., 2008) have dealt with the specification of a set of well-defined features desired for any RIA and the manner in which they should be developed using different model-driven methodologies. Nonetheless, the combination of Semantic Web techniques and technologies in the development of RIAs has not been studied in depth. Given that the Semantic Web technologies are specialised in representing and sharing knowledge across the Web, the combination of both approaches can solve the issues found in RIAs in a technology-independent manner, i.e., providing a solution that could be applied to any type of RIA.

In the light of these considerations, this chapter introduces a new class of RIA called *Semantic Rich Internet Application*, which extensively use Semantic Web techniques, technologies and resources for sharing their own data and reuse data from other sources to enrich their own contents.

The following sections describe all the aspects of this type of applications, starting with their requirements. Consequent to these requirements, the structure of the application is defined and explained, highlighting the software modules missing in traditional RIAs. Finally, this chapter introduces a collection of case studies used to evaluate the proposal and validate the fulfilment of the proposed requirements.

3.1 REQUIREMENTS

The definition of a specific set of requirements for a Web application facilitates the identification of the main goals and the main software components of the application. In this case, these requirements also show the differences between traditional RIAs and SRIAs. The requirements for the development of SRIA combine aspects of RIA and other aspects of Semantic Web applications in such a way that the resulting applications could be considered RIAs as well as Semantic Web applications.

This section proposes a specific list of requirements that characterise SRIAs focusing on those requirements that are not considered in the development of traditional RIAs. This list takes into consideration the studies carried out by other authors such as Brambilla and Facca (Brambilla and Facca, 2007) and Roval et al. (Rovan et al., 2011), focused on the engineering of Semantic Web applications, as well as the initial architecture of the Semantic Web and the principles of Linked Data, described in Section 2.1.2 (page 19).

The proposed features can be summarised in two high-level non-functional requirements. The fulfilment of these first requirements is associated to the fulfilment of a collection of functional requirements described below each, which constrain the functionalities of the resulting applications. The proposed list of requirements can be described as follows:

R1) High level of exportability and reusability of the application content. The application must be capable of providing its contents in a meaningful, unambiguous and structured manner to software agents or even other Semantic RIAs.

Rf1.1) The application must use ontologies as knowledge representation formalism. All the data stored and managed by a SRIA must be represented by means of ontologies, since they are the standard for the representation knowledge on the Semantic Web.

Rf1.2) The application must provide semantic annotations of the content. Ontologies provide a method to represent and structure the underlying knowledge used by a SRIA. However, it is also necessary to map the data stored in a SRIA into ontology instances and annotate certain chunks of information to effectively represent what information is being shown in a certain moment. This information will be contained in the annotation model (Bettencourt et al., 2006) proposed for SRIAs, which will be introduced in the next subsection.

R2) High level of reusability of external knowledge. Following the philosophy behind the Semantic Web and the principles of the Linking Open Data project, the application should be enriched with knowledge from other sources. The application should not be isolated, but be capable to obtain knowledge from different sources of the Semantic Web. This requirement can be split into two sub-requirements:

Rf2.1) The application must reuse existing ontologies. As a result, it would be possible to interconnect knowledge among a network of applications leading to richer user contents. Moreover, it can simplify the processes of knowledge sharing (R1) and the processes of design and development of similar applications.

Rf2.2) The application must reuse existing knowledge bases. With the instances obtained from other Web sources it would be possible to enrich the contents showed to users by means of mashups. Initially, in order to make easier the complexity of reusing knowledge in an open domain, only two types of knowledge sources are considered.

Rf2.2.1) The application must reuse knowledge from the available Linked Data sources/services. The application will use ontology instances from the Linked Data datasets spread across the Web.

Rf2.2.2) The application can reuse of knowledge from other applications. SRIAs should be compatible with other SRIAs, i.e., the knowledge shared by a SRIA should be consumed by other SRIAs following the Linked Data principles.

In this requirement list, ontologies and knowledge bases, which contain ontology instances, are considered as different elements, even thought, in several approaches, e.g., Gómez-Pérez et al. (Gómez-Pérez et al., 2007), instances are part of ontologies. Although it could be a controversial decision, it was taken based on the definitions given by Gruber (Gruber, 1995), and supported by Guarino (Guarino, 1998): "*an ontology serves a different purpose than a knowledge base state*". While the instances contained in the ontologies can be considered as shared knowledge describing a domain, the instances contained in a knowledge base "*may include the knowledge needed to solve a problem or answer arbitrary queries about a domain*". This approach facilitates the conceptualisation of those repositories of ontology instances with different goals that store instances of a common ontology, which is not an unusual case on the Semantic Web or the Web of Data.

3.2 An Annotation Model for Semantic Rich Internet Applications

The requirement *Rf1.2* establishes the necessity of an annotation model that defines the manner in which the application will be annotated and how the ontology instances will be shared. This section introduces the annotation model proposed for SRIAs based on the different annotation models available on the Semantic Web.

The proposed annotation model is based on ontologies (fulfilling *Rf1.1*). However, as mentioned before, ontologies can only provide the formalisms and mechanisms to capture and represent the underlying knowledge used by a SRIA. In some cases, it might be also relevant to annotate certain chunks of information to effectively represent what information is being shown to the users.

Before introducing the actual annotation model, in order to avoid ambiguities, the first step is to define the concept of semantic annotation in this context. Based on the definition by Bettencourt et al. (Bettencourt et al., 2006), a semantic annotation is a reference (i.e., a HTTP URI/IRI) to an ontology element (i.e., a concept, a property or an instance) that is attached to a resource (i.e., from a chunk of information or a complete SRIA). In "traditional" Semantic Web applications, these annotations can be shared within the HTML code of the user interface, which might attached to the resource annotated. However, since some types of RIA do not contain user interfaces with HTML code (plugin-oriented RIAs), it is necessary to provide alternative methods of accessing the annotations or the ontology instances.

Given these initial considerations, the SRIA annotation model proposed in this thesis contains the following elements:

1. The annotation model is composed of three types of ontology-based annotations divided in two groups: content and context annotations. *Content annotations* refer to the content of the application, which depends on the domain and the goals of the application. This group includes:

(i) domain annotations, which include information about the domain of the application. These annotations are related to a domain ontology (concepts, properties or instances) that might change from one application to another depending on the goals and type of content of the application. For instance, if the target application is a media player, the domain ontology will contain concepts related to the music domain, while, if it manages the information of a university, the domain ontology will contain concepts from the educational domain.

Context annotations include information related to the context where the main information is shown, e.g., the relative position within the Web application, the path followed to a certain internal state or Web page, the UI element in which the information is visualised, etc. This group includes the following types of annotation:

(ii) navigational annotations, which are associated to knowledge about the navigational aspects of the SRIA: navigational nodes, transitions, links, etc. These aspects are represented in a navigational ontology called *NavOntology*, which should be instantiated by any application. This ontology and its instances can be linked to any domain ontology and, thus, it will be possible to create a large network of ontology instances that will help to locate contents on the Web (Hermida et al., 2009). NavOntology was initially designed for traditional Web applications and subsequently adapted in order to capture the knowledge related to the navigation in RIA. For a detailed description of the elements of the ontology, please check Annex B.

(iii) visualisation annotations, which are related to knowledge about the structure (UI components) and behaviour of the visual elements of the application. These elements are represented by

means of a visualisation ontology, which captures knowledge about the components of the UI in terms of screenshots, widgets and panels (including their size, position, contents, aesthetics, etc.) and the events that can be triggered from each of them under certain conditions and the actions performed as a result. In this case, the ontology is also shared by all the SRIAs, which will need to instantiate it according to their UI. The elements of this ontology can be linked to the elements of NavOntology and the domain ontology of the application. For a more detailed description, please check Annex B.

Using this three-layer annotation model, it is possible to obtain/generate a complete representation of the data objects and the context in which they are visualised within the SRIA, which can be useful for better understanding the content of the application and automatically replicate the behaviour of the application in other platforms (e.g., mobile). Figure 3.1 illustrates the manner in which the three ontology representation can be combined and the resulting knowledge obtained. The figure depicts the instances of the three mentioned ontologies for a SRIA media player (please, see the description of the case study in Sections 3.4 and C.1).

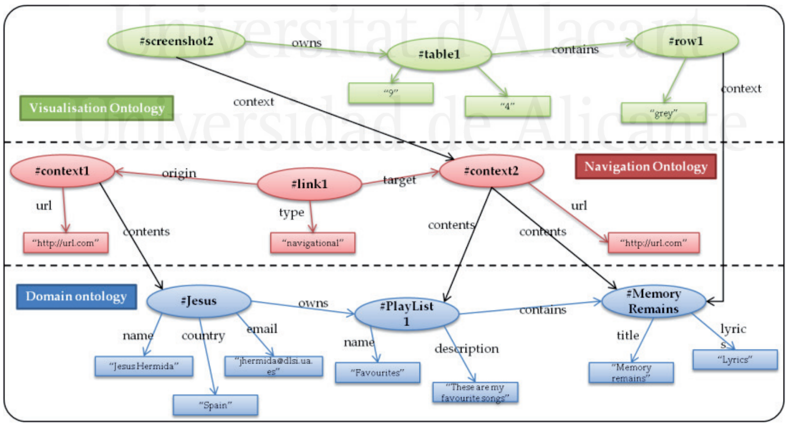


Figure 3.1. Example of instantiation of the three layer ontology representation for SRIAs.

The instances of the domain ontology are coloured in blue and show a certain user with their playlist and their music track. Coloured in red, the instances of the NavOntology specify the different navigation contexts and the domain instances shown in each of them. Finally, the instances of the visualisation ontology represent the different elements that can be visualised and the links to NavOntology and the domain ontology.

2. Embedded textual annotations. When developing browser-oriented SRIAs, textual annotations will be included following the three-layer representation and using the RDFa standard (World Wide Web Consortium, 2008a) within the text-based contents of the RIA (mainly HTML code).

3. Open service for sharing knowledge. For those SRIA in which it is not possible to embed textual annotations, i.e., plug-in-oriented SRIAs, it is necessary to provide another means of gaining access to the ontology representation of the contents. Opening a point of access to the contents, based on the efforts of the Linking Open Data Project[36] for data sharing and reuse, can simplify and boost the processes of knowledge retrieval (ontologies and instances) from a SRIA. In this context, the service most frequently implemented on the Semantic Web and the Linked Data cloud is the SPARQL endpoint[37]. Given that this solution does not cause any type of drawback, it could also be applied to browser-oriented as an alternative method of obtaining a representation of the application knowledge.

3.3 STRUCTURE

Once the requirements and the SRIA annotation model have been clarified, the next step is to define the structure of the SRIA, which represents the software modules of the SRIA and specifies their function. Figure 3.2 illustrates a schema of the structure proposed in this thesis. This schema also represents the processes of consuming knowledge from other sources, i.e., the Linked Data cloud and another SRIA.

Similarly to RIA, SRIAs are developed according to a client-server architecture whose clients, which contain rich UI interfaces, invoke the

[36] `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

[37] A definition of the service can be found at `http://sandbox.semantic-mediawiki.org/wiki/SPARQL_endpoint`: "*A SPARQL endpoint is a conformant SPARQL protocol service as defined in the SPROT specification. A SPARQL endpoint enables users (human or other) to query a knowledge base via the SPARQL language. Results are typically returned in one or more machine-processable formats. Therefore, a SPARQL endpoint is mostly conceived as a machine-friendly interface towards a knowledge base. Both the formulation of the queries and the human-readable presentation of the results should typically be implemented by the calling software, and not be done manually by human users.* "

Web services offered by the server using asynchronous communication processes using the Web infrastructure. In this way, SRIA clients and servers are totally decoupled.

The SRIAs server reuses part of original RIA components, specifically, those components that perform the basic operations over data:

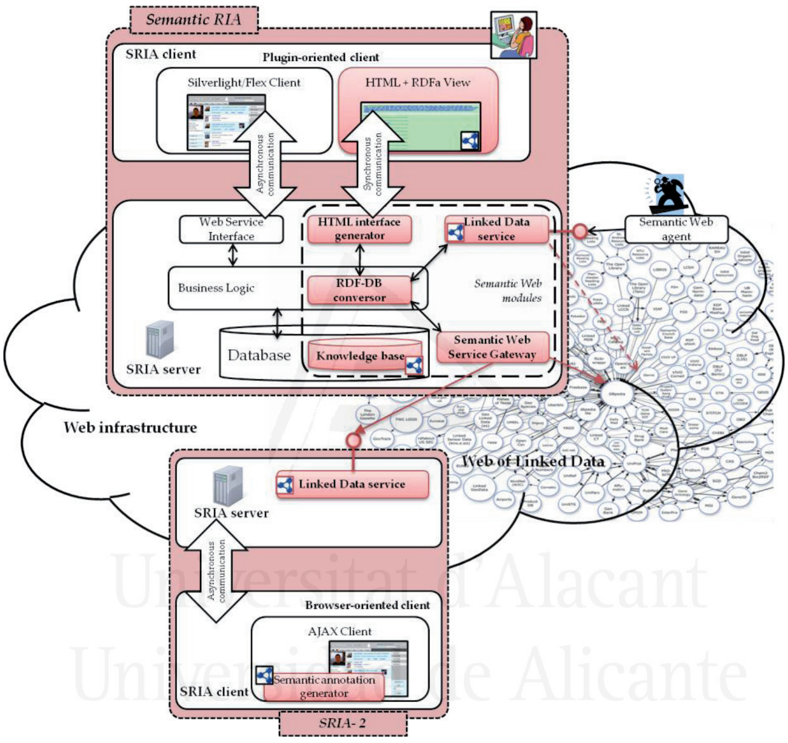


Figure 3.2. Schema of a semantic RIA and the relations with other elements of the Web[38]**.**

1) **Database**, which manages the persistent storage of the data objects.
2) **Business Logic**, including those components that perform the main tasks of the application and manage all the objects retrieved from the database.
3) **Web Service Interface**, which offers a set of server services to the UI interface that provides access to the server data and business logic.

[38] Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. `http://lod-cloud.net/`

Regarding SRIA clients, their UIs share most of the software components of RIA UIs. Depending on the technology employed to implement the RIA client, they can be classified into two categories: plug-in-oriented (Figure 3.2, SRIA) or browser-oriented (Figure 3.2, SRIA-2). The classification of RIA applications was explained in Section 2.1.1 (page 14).

As illustrated in Figure 3.2, apart from the RIA modules, SRIAs include a set of new software modules in order to fulfil the proposed requirements, directly related to the reuse of knowledge from the (Semantic) Web. Each of the new modules can be described as follows:

1) **Knowledge base (server module)**. This module manages the knowledge base (KB) of the application, which stores the ontology instances used in the application (based on RDF – *Resource Description Framework*). Given the need to reuse knowledge from the Semantic Web, SRIAs require as a storage system a KB, which can be implemented over the existing database.
2) **Linked Data service (server module)**. This module offers a service to access a part of the knowledge stored in the SRIA knowledge base. In this case, this approach is aligned to Linked Data principles following the SPARQL protocol for RDF (World Wide Web Consortium, 2008b). Nonetheless, this interface could be changed depending on the requirements of the application by other type of service, e.g., semantic Web services (SWS). Since the structure of the query can directly affect the performance of the service, in this proposal, the SPARQL endpoint can limit the access to a certain number of classes or instances of a class depending on the developers' preferences.
3) **Semantic Web service gateway (server module)**. This gateway actually groups several types of service clients: Web services (SOA, REST), semantic Web services and Linked Data endpoints. This module enables the access to ontologies and KBs on the Web on demand (even those of other SRIAs) in order to enrich/complete the content provided to the end users. Depending on the resource to be accessed, the gateway chooses one access method, e.g., for Linked Data endpoints, it uses the SPARQL protocol by means of a SPARQL-enabled client.

4) **Semantic annotation generator (client module, browser-oriented interfaces)**. In browser-oriented SRIAs, the client can include a software module that embeds RDFa annotations, which links the shown UI with the ontology instances stored in the KB or external knowledge sources. In the case of plug-in-oriented SRIAs, users could access the same knowledge through the Linked Data service or the **HTML+RDFa view** generated by the following module.
5) **HTML interface generator (server module)**. This module generates an HTML representation of the ontology instances stored in the KB. This view is annotated using RDFa referencing the ontology instances stored in the KB. Unlike RIA UIs, this interface can be easily crawled and indexed by Web searchers. The entry point to the interface is a URL included in the header of the header of the HTML Web page containing the RIA client. In this UI, the communication between the client and the server follows the synchronous process used in traditional Web applications.

It is worth noticing that the manner in which SRIAs produce and consume data/knowledge from/to the Semantic Web is based on the existing Semantic Web applications, and more specifically, on the applications that interact with the Linked data cloud. From the Linked Data perspective and as a consequence of the defined structure, SRIAs can be treated as new nodes of the Linked Data network (as Figure 3.2 intends to represent). In the case of plug-in-oriented SRIAs, users would still be able to access the knowledge of the application through the Linked Data service. The annotated HTML view of the interface can facilitate the access to data to those clients that are not adapted to the Linked Data cloud.

This general structure aims to be a guide to the components that are needed for the development of SRIAs. Based on this, it would be possible to choose different architectural patterns for the development of different components of the SRIA. For instance, the SRIA client could be developed using different patterns, e.g., *Model-View-Controller*, *Model-View-Presenter*, *Model-View-Viewmodel*, etc. The architecture chosen in this thesis is introduced in Section 7.1 (page 152).

3.4 Case Studies

Semantic RIAs can be used a generic platform for the development of different types of applications, since they bring a technology-independent solution, using existing techniques, to solve the main shortcomings of RIAs. The assessment of the SRIA proposal was performed by means of the development of a set of case studies. The qualitative analysis of the applications developed helped to improve the approach in an iterative process. Each of these case studies was also validated externally in national and international conferences and international journals. This chapter introduces three case studies, developed in this evaluation process:

1) The development of a media player, inspired by the case study presented by Brambilla and Facca (Brambilla and Facca, 2007), which aims at building a SRIA as a media player for the management of the users' personal songs. The application can share the personal data about music files as instances of the MusicOntology ontology and can reuse the information published by the MusicBrainz Linked Data endpoint.
   This case study was first introduced by Hermida et al. (Hermida et al., 2011b) and it is fully explained in Annex C.
2) The development of a social network site on the Semantic Web, as defined by Kinsella et al. (Kinsella et al., 2009), which aims at developing a SRIA as social network site for sharing knowledge/comments about music. This case study was first introduced by Hermida et al. (Hermida et al., 2011a) and the details of this case study will be explained in the next subsection, since it will be used as a reference in the rest of the thesis.
3) The development of a SRIA for Business Intelligence. The SRIA approach was also applied in the field of Business Intelligence, leading to the development of a new case study. This last case study consisted in the development of a social network site for sharing knowledge among the employees of a company. Given the special requirements detected in this last case study, it will be explained in Section 3.5.

The case studies were developed using the .NET framework (C#) and, particularly, the Windows Communication Foundation (WCF, for the server components) and Silverlight (for the client components) frameworks.

3.4.1 A Social Network Site as a Semantic RIA

RIAs and Social Network Sites share some shortcomings that mainly limit the portability of the data stored. Social sites are normally born as proprietary sites where their API-based access methods do not share all their available information and the semantics of the data elements might vary between applications. Currently, it is usual that users have different profiles in several networks since they cannot reuse their own personal data (Breslin and Decker, 2007; Breslin et al., 2009). As mentioned before, RIAs show the information in a user-friendly manner but, due to their intrinsic structure and technological issues, they also suffer these types of limitations. The application of the SRIA approach to this case can help to solve the issues found in the social sites as well.

In this case, Figure 3.3, the schema of the SRIA when applied to the development of a Social Network Site slightly modifies from the original one including the representation of other applications of the Social Semantic Web (Kinsella et al., 2009), which could be accessed by the SRIA in order to retrieve knowledge from them (e.g., a semantic wiki). As can be appreciated, the structure of the SRIA is not modified.

Given the functionalities of the SRIA, the main benefits of applying the proposal to the development of social platforms are the following (Hermida et al., 2011a):

1) Benefits from the use of Semantic Web technologies:
    a. *Improved interoperability among social sites*. The use of ontologies such as FOAF or SIOC, allows Web designers to describe all the personal data of the SNS (list of friends, contents from the wall, etc.) in a standard manner and facilitates the process of sharing among different social sites. While the FOAF ontology is widely used on the Web to describe people, documents and their relations, SIOC can complementarily describe all the types of possible contents of a social site.

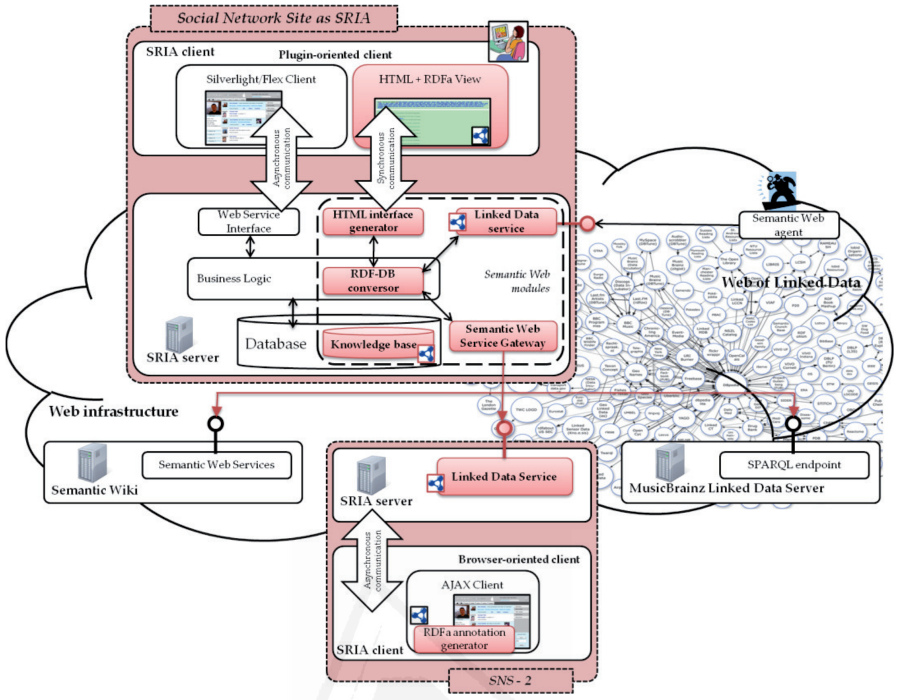


Figure 3.3. Schema of a social network site as a SRIA and the connections to other applications.

b. *Enrichment of the contents*. SRIAs can reuse knowledge from other sites or existing knowledge sources in order to enrich the content presented to users. For instance, a SRIA can automatically access knowledge contained in DBpedia (`http://www.dbpedia.org`) to obtain supplementary information about a certain topic. Moreover, depending on the RIA technology, SRIAs can add semantic annotations to the content visualised by the users, which can be employed by different Web users, such as searchers or special clients for people with physical impairments.

2) Benefits from the use of RIA technologies

a. *Improved usability of the user interface*. The use of a RIA interface in a SNS is not a novel approach. There currently exist some examples, such as *facedeck*[39], that offer a higher degree of interactivity to users and a friendlier, desktop-like UI. The main difference is that this approach aims at

[39] `http://www.telerik.com/products/facedeck.aspx`

building the complete SNS as a SRIA, not only the UI, thus improving the behaviour of the whole application.

The second use case is the development of a Social Network Site using a SRIA as a platform (Hermida et al., 2011a). The application will be focused on the music domain (instead of having a general purpose) and will embed a music player, similar to the one developed in the first case study, thus offering a platform for online music sales as well.

The present case study does not introduce new social features, which have been exploited by existing SNSs such as Apple's Ping[40] or lastFM[41]. Instead, it is aimed at studying the interconnection (interoperability) between social network sites using open SW techniques and resources, already studied by other authors, e.g., Kinsella et al. (Kinsella et al., 2009), and Rich Internet Applications.

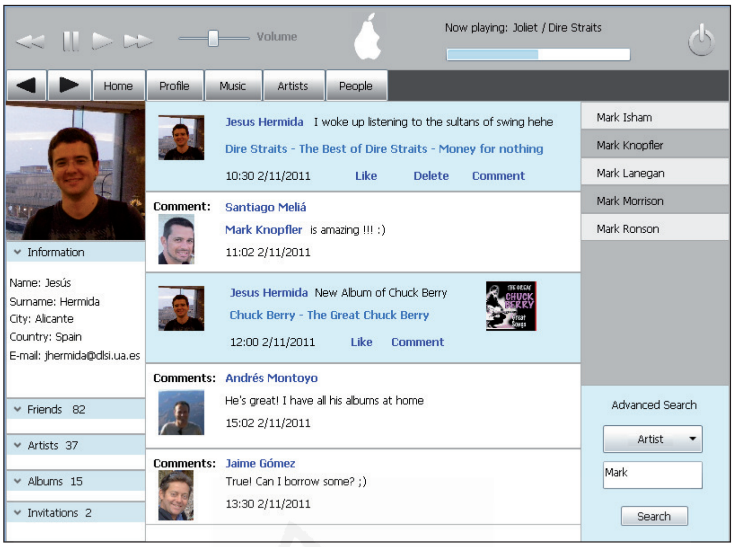
The final application will manage the basic features of current SNSs, i.e. *(a)* the management of a user profile, *(b)* the connections with other users and *(c)* a personal wall where users can share their thoughts, impressions or comments with their contacts. Moreover, users will be able to *(i)* share their music preferences, groups and songs; *(ii)* follow their favourite artists; and *(iii)* read and publish news on all these topics.

Figure 3.4 depicts a screenshot of the final application, whose UI can be divided in four different areas: *(1)* the music player, located on the top of the application, together with the main menu; *(2)* the user profile, right area, with the main user information; *(3)* the wall, in the central area, with the main user's and user friend's stories and comments; and *(4)* a search form, on the left part of the UI, with a form to search friends, artists or songs within the SNS.

Figure 3.5 shows the entry point of the HTML view for this application. From this Web site, users can visualise the data of different data entities by using one of the available links (i.e., MusicArtist, for music artists; Person, for users; Record, for music albums; or Track, for music tracks). Moreover, they can find the URI of the Linked Data service, which provides the data objects as ontology instances.

The final implementation of this application can be fount at `http://suma2.dlsi.ua.es/ooh4ria/sm4ria.html#uc`.

[40] `http://www.apple.com/itunes/ping/`
[41] `http://www.last.fm/`



Name: Jesús
Surname: Hermida
City: Alicante
Country: Spain
E-mail: jhermida@dlsi.ua.es

Jesus Hermida I woke up listening to the sultans of swing hehe
Dire Straits - The Best of Dire Straits - Money for nothing
10:30 2/11/2011

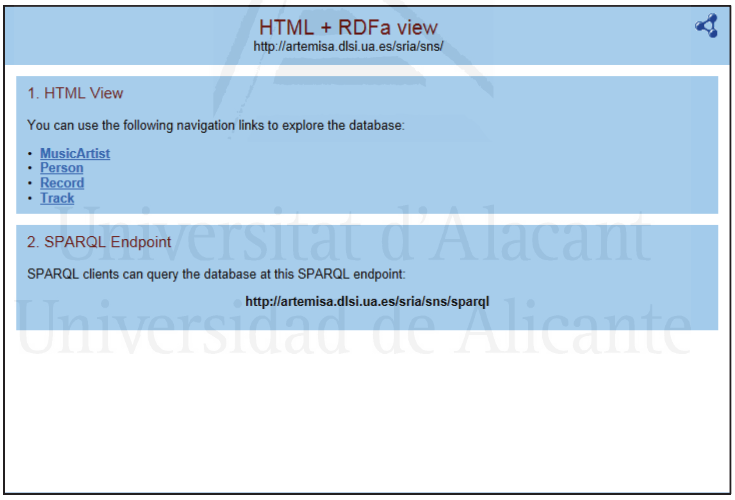
Comment: Santiago Meliá
Mark Knopfler is amazing !!! :)
11:02 2/11/2011

Jesus Hermida New Album of Chuck Berry
Chuck Berry - The Great Chuck Berry
12:00 2/11/2011

Comments: Andrés Montoyo
He's great! I have all his albums at home
15:02 2/11/2011

Comments: Jaime Gómez
True! Can I borrow some? ;)
13:30 2/11/2011

Figure 3.4. Screenshot of the main context of the Social Network Site.


HTML + RDFa view
http://artemisa.dlsi.ua.es/sria/sns/

1. HTML View

You can use the following navigation links to explore the database:

- MusicArtist
- Person
- Record
- Track

2. SPARQL Endpoint

SPARQL clients can query the database at this SPARQL endpoint:

http://artemisa.dlsi.ua.es/sria/sns/sparql

Figure 3.5. Screenshot of the HTML view for the Social Network Site.

3.5 Semantic Rich Internet Applications as platforms for Business Intelligence

As shown by Raspal (Raspal, 2010), White (White, 2009) or Laurent (Laurent, 2010), Rich Internet Applications (RIA) start to play a relevant role as Web applications for Business Intelligence due to their intrinsic features: UIs with high level of interactivity and usability (in contrast to traditional Web applications) and the asynchronous communication processes between the server and the client of a Web application, led by a set of mature technologies e.g., Flex/Flash, Silverlight or AJAX. RIAs have proven to be an appropriate platform to visualise data from different sources, whose rich user interfaces offer experiences similar to the ones provided by desktop interfaces.

However, RIAs are not as specialised in the management of knowledge from the Web as BI systems require. RIAs for Business Intelligence should deal with issues of knowledge management (KM) and visualisation that finally require complex software modules combined in a single architecture.

In order to explore the possibilities of SRIAs as knowledge management applications in business environments, in the last case study, the SRIA proposal was adapted to the BI environment, obtaining as a result a new subtype of application, the RI@BI (RIA for BI) (Hermida et al., 2013). This application overcomes the limitations of traditional RIA platforms applying knowledge management techniques from the Semantic Web and a collection of B2B components to improve the communication with other business Web applications.

The RI@BI proposal is based on the functionalities of the SRIA solution. Unlike SRIAs, RI@BIs aims at providing a solution adapted to the business scenario and thus considers aspects related to B2B (or SaaS) services, which have special relevance in this scenario.

3.5.1 REQUIREMENTS

Given the new needs and the known limitations of RIAs when applied to business scenarios, it is important to extend the set of requirements defined in Section 3.1 for SRIAs in order to include new functionalities that finally overcome them. These new, specific functional requirements can be described as follows:

Rf3) Consumption of B2B data/services from other departments/enterprises. The application should be connected to other departments'/enterprises' applications to reuse their data and/or services. Due to their growing importance, it would be especially relevant to support the use of SaaS services.

Rf4) Provenance of data/services to other departments/enterprises. As the application needs of data/services from other applications, it should be capable of providing these to other applications as well.

Rf5) Complex visualisations of data/knowledge (data mash-ups). The application should use the advance mechanisms for the integration of knowledge from different sources in a manner transparent to users. Not only should the visualisation components (widgets) represent data from the applications itself but also they should mix data/knowledge from different sources and represent them in complex widgets.

Within the Semantic Web, the approaches based on the concept of Linked Data offer a more ambitious perspective towards the creation of open enterprises or networks of enterprises that seamlessly share their data following a common goal. This approach is totally aligned to the SRIA approach and to the RI@BI approach, described in this section, as well.

Based on the structural schema of a SRIA, Figure 3.6 illustrates the schema of a typical RI@BI and the connections with other sources from the Web and other B2B services provided by other applications.

As shown in the figure, RI@BIs include a set of new software modules (coloured in green), which are directly associated to the exploitation of B2B services, in order to fulfil the proposed requirements.

1) **Service interface (server module).** This interface publishes a set of B2B services of the RI@BI to external organisations depending on the goals of the application. The application could provide different types of services: from data providers to data processing, or even SaaS services. Unlike the Linked Data service (e.g., an SPARQL endpoint), this service could handle secure accesses.
2) **Web service gateway (server module).** This module allows applications to invoke external B2B services provided by other organisations. For instance, it would be possible to access the private data of other enterprises or use external services to store the application data (e.g., Dropbox[42]).

[42] http://www.dropbox.com/

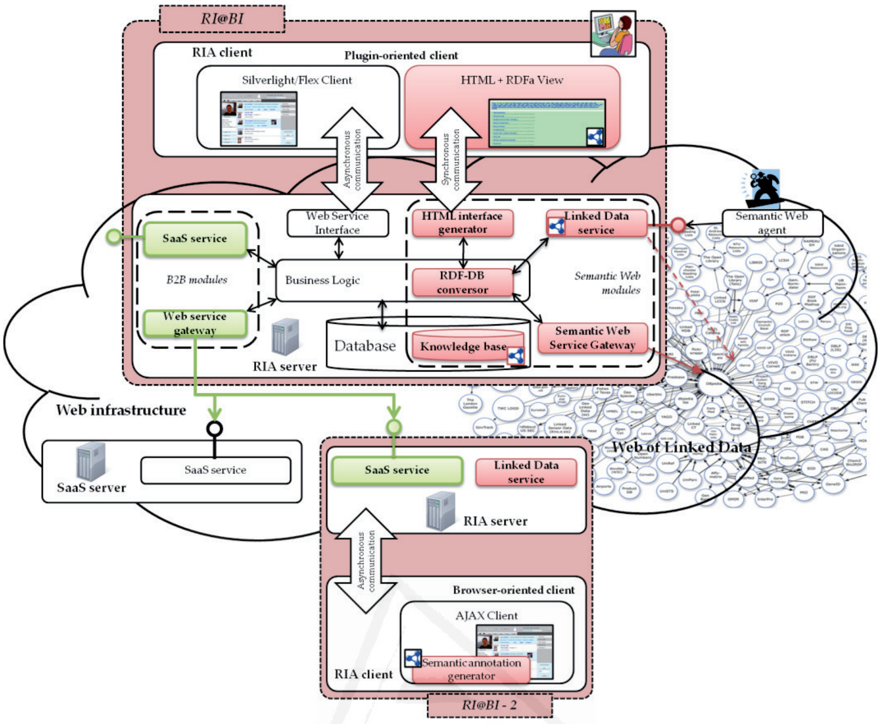


Figure 3.6. Schema of a RI@BI and the connections between the application and other applications and services.

3.5.2 Case Study: Developing a Social Application for Managing Business Knowledge

The case study chosen to assess this proposal concerns the development of a social network site (SNS) for the management of the knowledge related to the projects and the daily tasks of the employees of a specific enterprise using a RI@BI as a platform. This case study is based on the *miKrow* system (Penela et al., 2011) and the case study presented in Section 3.4.1. The target enterprise in this case is a software development company. The RI@BI includes the most usual, basic features of current SNSs:

(i) user profile and role;

(ii) different types of social connections;

(iii) a wall where employees can share their thoughts, impressions or comments with other users.

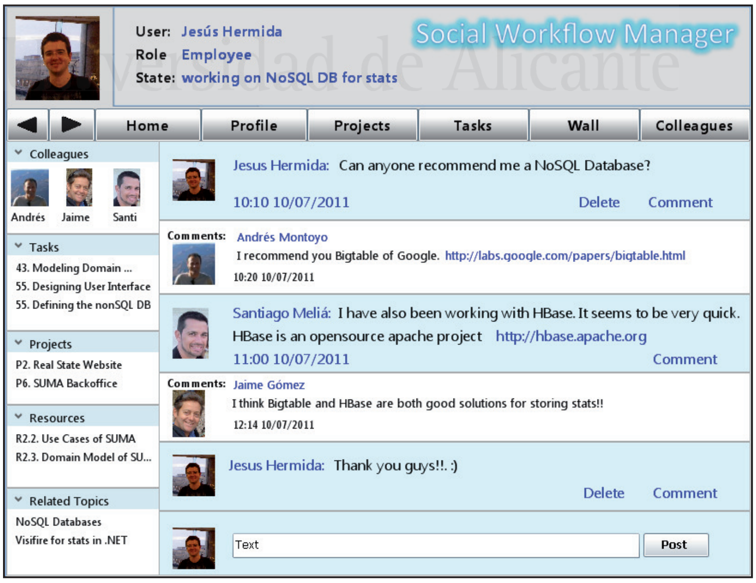
Moreover, given the purpose of the application, users will be able to:

a) create new tasks (only managers);

b) check the projects he/she is involved in;
c) check the tasks he/she is working on;
d) manage the resources (e.g., documents) associated to each project/task;
e) follow the activities of other users (only managers); and
f) (automatically) associate the content to ontology elements. This last feature will help users find other tasks, messages or colleagues that are involved in the same types of issues.

The aim of this case study is to analyse: *(i)* the capabilities of RI@BI as BI Web applications; and *(ii)* the interconnection (interoperability) between different business applications and knowledge sources, in particular, Semantic Web resources. As in the SNS case study, the final application does not include new social features, which have already been studied and exploited by other authors such as Yammer (2011), Penela et al (2011) and Kinsella et al (2009). Moreover, security and trust issues are out of the scope of this example.

Figure 3.7 contains a screenshot of the final application, which shows the last stories created by a user and the replies he received from his colleagues. The final Web application can be found at `http://suma2.dlsi.ua.es/ooh4ria/sm4ria.html#uc`.



User: Jesús Hermida
Role Employee
State: working on NoSQL DB for stats

43. Modeling Domain ...
55. Designing User Interface
55. Defining the nonSQL DB

P2. Real State Website
P6. SUMA Backoffice

R2.2. Use Cases of SUMA
R2.3. Domain Model of SU...

NoSQL Databases
Visifire for stats in .NET

Jesus Hermida: Can anyone recommend me a NoSQL Database?
10:10 10/07/2011

Comments: Andrés Montoyo
I recommend you Bigtable of Google. http://labs.google.com/papers/bigtable.html
10:20 10/07/2011

Santiago Meliá: I have also been working with HBase. It seems to be very quick. HBase is an opensource apache project http://hbase.apache.org
11:00 10/07/2011

Comments: Jaime Gómez
I think Bigtable and HBase are both good solutions for storing stats!!
12:14 10/07/2011

Jesus Hermida: Thank you guys!!. :)

Text

Figure 3.7. Screenshot of the project management application as a RI@BI.

3.6 CONCLUSIONS

The content of RIAs cannot be easily crawled and indexed by Web search engines, which can prevent developers from implementing RIA applications and enterprises from using this type of application for their business. In this context, the first contribution of this thesis is **the definition of the concept of Semantic RIA**, whose main goal is to provide a generic solution to this problem based on the existing Semantic Web techniques and independent from the implementation technology. The SRIA approach extends the structure of the traditional RIA including modules for sharing local knowledge and consuming knowledge from the Semantic Web. The knowledge managed by the SRIA is completely available as linked data using the annotation model proposed. This three-layer model combines different existing approaches and can provide a complete view of the application from the user perspective, hiding design or implementation details, in such a way that this knowledge could be reused to create widgets for other Web or desktop applications.

The three case studies developed have been used to assess the proposal in three different scenarios: *a)* the Semantic Web with the development of a media player, *b)* the Social Semantic Web with the development of a SNS, and *c)* the field of Business Intelligence with the development of the enterprise SNS. The qualitative assessment was performed either internally, i.e., by analysing the features developed and the possible improvements, and externally, i.e., in international conferences and journals. The evaluation of performance and scalability issues was not in the scope of the analysis carried out.

The complexity of the resulting SRIA architecture notably increased the cost of development and maintenance (comparing to RIA or traditional Web applications), which is a risk factor that threatens the viability of this type of applications and their future success, especially when working in business scenarios. SRIAs require complex software modules combined in a single architecture in order to deal with issues of knowledge management and visualisation.

Chapter 4. A METHODOLOGY FOR THE DEVELOPMENT OF SEMANTIC RICH INTERNET APPLICATIONS

During the past decade, model-driven methodologies have been proven to successfully address all the phases in the development of different types of Web applications, including RIA. This type of methodologies (usually supported by CASE tools) facilitates the systematic design and generation of Web applications and can be an appropriate solution for the development of SRIA, given the complexity of their structure. Model-driven methodologies can reduce the cost of development in terms of time and resources and, thus, can minimise the risk of project failure, which are important factors when developing applications in the business context.

These methodologies are relatively new and one of the aspects not yet supported is the development of RIAs capable of managing information from the Semantic Web. In order to address the development of SRIAs, this thesis presents the S^{m}4RIA methodology[43] on the basis of the OOH4RIA methodology (Meliá et al., 2008), specialised in the development of traditional RIA. The goal of this methodology is to cover all the phases of development of SRIAs from the design of the data entities and the user interface to the generation of the software modules.

[43] Semantic Models for RIA, /*sem for RIA*/. Originally introduced by Hermida et al. (Hermida et al., 2011b)

In order to fulfil the requirements of SRIAs, S^{m}4RIA defines new processes and artefacts:

(i) two new MOF metamodels, created as an extension of the OMG Ontology Definition Metamodel (Object Modeling Group, 2009), which lead to the definition of two new ontology models;

(ii) a set of model-to-model transformations that can create mock-ups of the different S^{m}4RIA models, which can help the designer to obtain an ontology model from another;

(iii) a set of model-to-text transformations that generate the new SRIA software modules from the whole collection of models.

Furthermore, the S^{m}4RIA methodology extends the OOH4RIA development process:

(i) including new modelling mechanisms to the OOH4RIA functional models as an extension of the OOH4RIA MOF metamodel.

(ii) adapting the existing activities to the development of SRIAs by adding new tasks and modifying the existing ones.

In order to contextualise the contribution of S^{m}4RIA, the next section introduces the main process and components of the OOH4RIA methodology. Subsequently, the chapter will introduce all the aspects of the S^{m}4RIA methodology.

4.1 An Introduction to OOH4RIA

OOH4RIA (Meliá et al., 2008) defines a model-driven methodology whose main aim is to cover all the phases of the RIA lifecycle development. It facilitates the specification of a RIA (all its software components, whose final code can be partially personalised by the developers) by means of four models: Domain and Navigation models, extending the OO-H server-side models (Gómez et al., 2001), and two RIA-specific presentation models, i.e., the Presentation and Orchestration models.

The OOH4RIA methodology defines a development process for RIA that starts with the specification of the Domain model, which represents the data structures, the relations between them and the operations that can be performed. The Domain meta-model allows designers to specify the Object-Relational mapping rules without ambiguities.

With the OOH4RIA Domain model, designers can *(1)* classify the class operations based on a predefined topology (including elements such as *create*, *delete*, *relationer*, *unrelationer*, etc.) to facilitate the generation of the CRUD functionality of a data-intensive application server; *(2)* use a collection of complex data types such as *set*, *bag* or *list*, for defining class attributes and operations; and *(3)* define Object-Relational mapping using concepts not considered in OO-H such as the *object identifier* (used for defining the primary key).

After the Domain Model, the designer specifies the Navigation Model, which defines the navigation of the user through the application data and the invocation of the class operations. This model establishes the most relevant semantic paths through the data space, filtering the domain elements that will be available in the RIA clients. In addition, it facilitates the definition of OCL filters for gathering information from the domain classes.

At this point, the UI designer defines the structure (widgets, panels and style) and the behaviour of the RIA client (and rich user interface) using the Presentation and Orchestration model. At present, there are several RIA frameworks, each of which offers different sets of widgets, properties and events. The OOH4RIA Presentation model is a platform-specific model which is adapted to the elements of the Silverlight framework, even though it can be adapted to other technologies. This model, due to its graphical notation (WYSIWYG), offers a visual representation to the designers, similar to the final RIA UI that will be generated. To complete the specification of the RIA UI, the OOH4RIA Orchestration model can be used to describe the behaviour of the components of the RIA UI by means of the definition of a set of Event-Condition-Action (ECA) rules over the UI widgets.

Once the models have been defined, in the final activity, a set of model-to-text transformation rules can be invoked in order to generate the RIA implementation from the information stored in the models. OOH4RIA defines two main transformation processes: one for the generation of the RIA server from the Domain and Navigation models; and another for the generation of the RIA client side from the Presentation and Orchestration models.

From this description the following sections explain the main elements and activities of the S^{m}4RIA methodology.

4.2 Semantic Models for Rich Internet Applications

Figure 4.1 shows an overview of the S^{m}4RIA development process using an SPEM2[44] class diagram, conformant to an extension of the SPEM2 meta-model that includes aspects not considered in the original meta-model, e.g., the representation of the transformation engines (called *Model Transformers* in the model) using a new stereotype called *ProcessRole* and the MDA transformations using a new collection of stereotypes extending the *TaskDefinition* SPEM meta-class, e.g., *PIM2PIM, PIM2PSM, etc*.

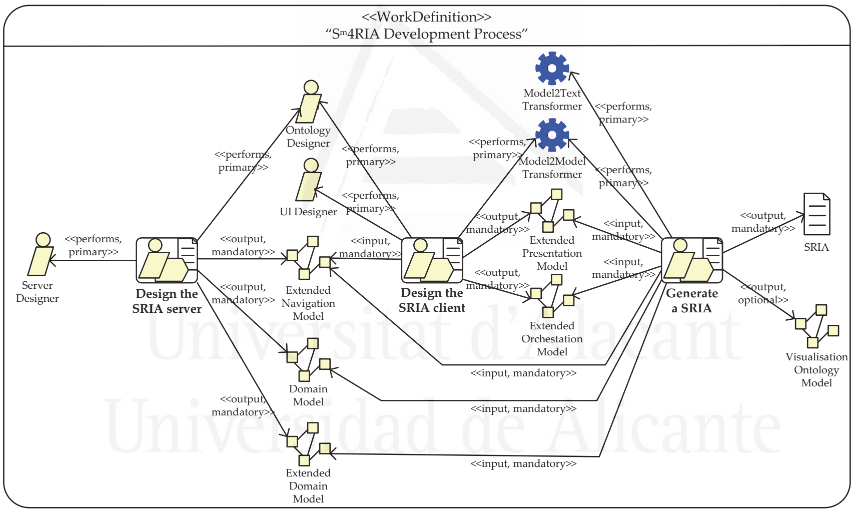


Figure 4.1. SPEM2 class diagram of the S^{m}4RIA development process.

The following subsections explain each of the components illustrated in the figure, i.e., the user roles and the models (including the meta-models), and the development process as a whole. The final result of the process, i.e., the SRIA, has already been explained in Chapter 3.

4.2.1 The S^{M}4RIA User Roles

There are five types of user role involved in the different activities of the process (see Figure 4.1):

[44] Software Process Engineering Meta-model (Object Management Group, 2008).

a. **Server designer or back-end designer.** The server designer creates the server components of the SRIA, e.g., the database, the Web service interface, etc.
b. **User-interface (UI) designer.** The user interface designer performs those tasks related to the construction of the SRIA user interface and the invocation of the services provided by the SRIA server.
c. **Ontology designer.** The ontology designer carries out the tasks related to the interconnection of the SRIA with external knowledge sources. These tasks could be also performed by the Server designer depending on their personal background.
d. **Model-2-model transformer.** This role corresponds to the transformation engine capable of transforming one model into another.
e. **Model-2-text transformer.** This last role corresponds to the transformation engine capable of transforming the content of a model into programming code.

4.2.2 THE S^M4RIA MODELS

There are six models involved in the development process of a SRIA, which address different concerns:

1. **Domain Model (*Platform-Independent Model*).** The Domain model, imported from the OOH4RIA methodology with no modification, and defines the main data structures of the application (classes, attributes and types), the relationships among them and the operations that can be performed over them. The operations are classified into two groups: CRUD operations (create-read-update-delete), which are the basic operations over any data object and the custom operations, whose signature can be defined by the designer. Furthermore, it defines the Object-Relational mappings that will be used to transform the relational database tuples into data objects.
2. **Extended Domain Model (EDM, *Platform-Independent Model*).** The EDM was created for the definition of lightweight ontologies that could represent the domain entities of the application and the relationships among them. Moreover, this model captures the

ontologies imported from other sites and the knowledge bases available for each of the ontologies modelled. The specific goals of the model can be described as follows:

   a. represent the domain ontology of the application;
   b. establish the relations between the SRIA ontology and external ontologies, thus aligning the domain elements with external elements (or even reuse external elements in local ontologies);
   c. define the external sources that will be available to the SRIA users;
   d. define the mapping rules between the ontology elements and the data objects defined in the Domain Model;
   e. define operations over ontology instances (e.g., for filtered searches of external instances).

3. **Extended Navigation Model (ENM, *Platform-Independent Model*).** The Extended Navigation Model is an extension of the OOH4RIA Navigation Model. The ENM specifies the manner in which users will be able to access the data and the ontology instances defined in the last two models. For each user role, it is possible to define a different navigational model that filters the information retrieved from the SRIA server and the services that can be invoked. The ENM also captures the manner in which SRIAs publish their own structured knowledge and connect their information to other sources of knowledge on the Web.
4. **Extended Presentation Model (EPM, *Platform-Specific Model*).** The EPM extends the OOH4RIA Presentation Model. This model specifies the structure of the SRIA representing the screenshots, panels and widgets of the UI as well as their main features: position, size and style (text font, font colour, background colour, etc.) In contrast to the other models, this is a WYSIWYG model, in which the visualisation of the UI model should be completely equivalent to the generated UI. In this model, it is also possible to include ontology-based annotations over static UI components.
5. **Extended Orchestration Model (EOM, *Platform-Specific Model*).** The EOM is an extension of the OOH4RIA Orchestration model. The EOM is represented as a collection of Event-Condition-Action rules that specifies the behaviour of the user

interface according to the events triggered by the users when interacting with the UI elements. It connects the UI events with the actions that can be performed by the SRIA server, which are specified in the EDM and ENM.

6. **Visualisation Ontology Model (VOM, *Platform-Independent Model*).** The Visualisation Ontology model combines the knowledge contained in the EPM and EOM in order to create the instances of the Visualisation Ontology needed for SRIA annotation model. This model should be automatically built using a M2M transformation.

4.2.3 THE S^{M}4RIA DEVELOPMENT PROCESS

In a similar manner that OOH4RIA, the S^{m}4RIA development process is divided in three main activities, which group tasks and modelling elements with the same final goal:

1. To design the elements of the SRIA server;
2. To design the elements of the SRIA client; and
3. To generate the final SRIA through a set of automatic model-to-text transformation processes.

The SPEM2 sequence diagram of each of the activities is illustrated in Figure 4.2, Figure 4.3 and Figure 4.4. Those tasks coloured in orange are new tasks or have been reused and modified from OOH4RIA.

The S^{m}4RIA development process starts with the design of the server. In this activity, the designers model all the aspects that will be used during the process of generation. The first task, performed by the Server designer, is the definition of the Domain model, which defines the main data structures of the application, the relationships among them and the operations that can be performed over them. Subsequently, the Ontology designer creates the EDM, which builds the domain ontology, imports external ontologies and KBs and maps the ontology instances and the data objects of the SRIA.

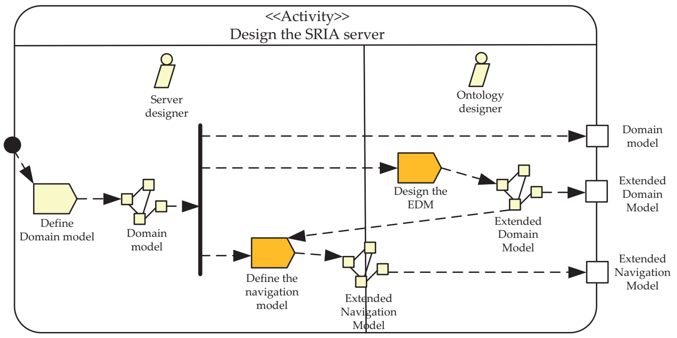


Figure 4.2. First activity of the S^{m}4RIA development process.

Both domain models are the input of the *Define the Navigation Model* task, in which the server designer specifies the manner in which users will be able to navigate across the data and the ontology instances (local or external). Moreover, it defines which server operations will be invoked by the UI or by external clients. This model also represents the invocation of external services from the Semantic Web (e.g., SWS or Linked Data endpoints), previously defined in the EDM.

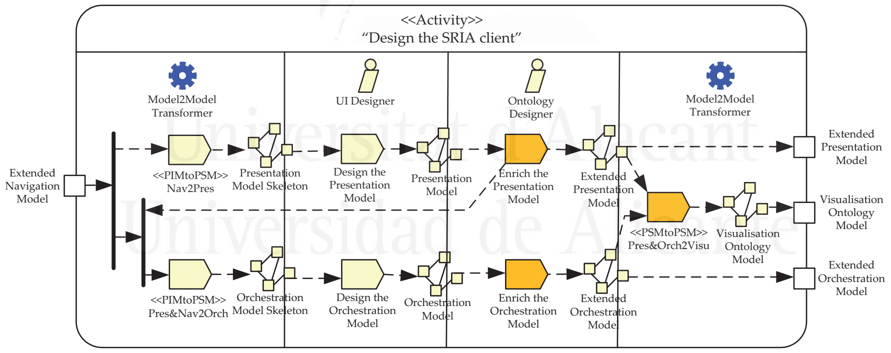

<<Activity>>
"Design the SRIA client"
Model2Model Transformer
UI Designer
Ontology Designer
Model2Model Transformer
Extended Navigation Model
<<PIMtoPSM>> Nav2Pres
Presentation Model Skeleton
Design the Presentation Model
Presentation Model
Enrich the Presentation Model
Extended Presentation Model
<<PSMtoPSM>> Pres&Orch2Visu
Visualisation Ontology Model
<<PIMtoPSM>> Pres&Nav2Orch
Orchestration Model Skeleton
Design the Orchestration Model
Orchestration Model
Enrich the Orchestration Model
Extended Orchestration Model
Extended Presentation Model
Visualisation Ontology Model
Extended Orchestration Model

Figure 4.3. Second activity of the S^{m}4RIA development process.

The second activity, i.e., the design of the SRIA client or user interface, continues by transforming the Extended Navigation model into the EPM and, subsequently, the EOM, through two model-to-model transformations called *Nav2Pres* and *Nav&Pres2Orch*. The resulting models should be completed by the UI designer. It is worth noticing that these transformations are optional since the UI designer can also create the EPM and EOM from scratch. Additionally, the *Enrich Presentation Model* and *Enrich Orchestration model* tasks can also be performed by the UI designer. In these two tasks, the ontology designer includes the static ontology-based annotations and links the UI elements to the external knowledge sources. Once the EPM and the EOM are completed, they are both transformed into the Visualisation Ontology Model using the *Pres&Orch2Visu* M2M transformation, thus combining and abstracting the knowledge captured in them.

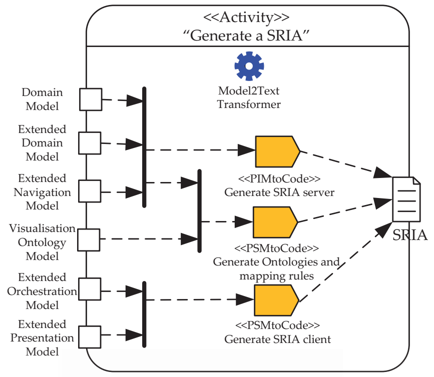

<<Activity>>
"Generate a SRIA"
Model2Text Transformer
Domain Model
Extended Domain Model
Extended Navigation Model
Visualisation Ontology Model
Extended Orchestration Model
Extended Presentation Model
<<PIMtoCode>> Generate SRIA server
<<PSMtoCode>> Generate Ontologies and mapping rules
<<PSMtoCode>> Generate SRIA client
SRIA

Figure 4.4. Third activity of the S^{m}4RIA development process.

Finally, in the last activity the final SRIA software modules are automatically generated from the set of models created in the first two activities through a set of model-to-text (*PSM2Code)* transformations. These transformation processes cannot generate all the code of the software components, e.g., customised operations cannot be automatically generated. Part of the code should be manually completed by developers.

The whole development process will be explained in depth in the next chapters using the SRIA case studies. In this way, the validity of the S^{m}4RIA approach could be assessed qualitatively.

4.2.4 The S^{M}4RIA Metamodel (Concrete/Abstract Syntax)

The S^{m}4RIA models introduced in Section 4.2.2 are conformant to the S^{m}4RIA meta-model, which extends the OOH4RIA meta-model and the OMG's ODM. Figure 4.5 depicts an UML package diagram with the S^{m}4RIA meta-model, its components and the links to the meta-models they extend.

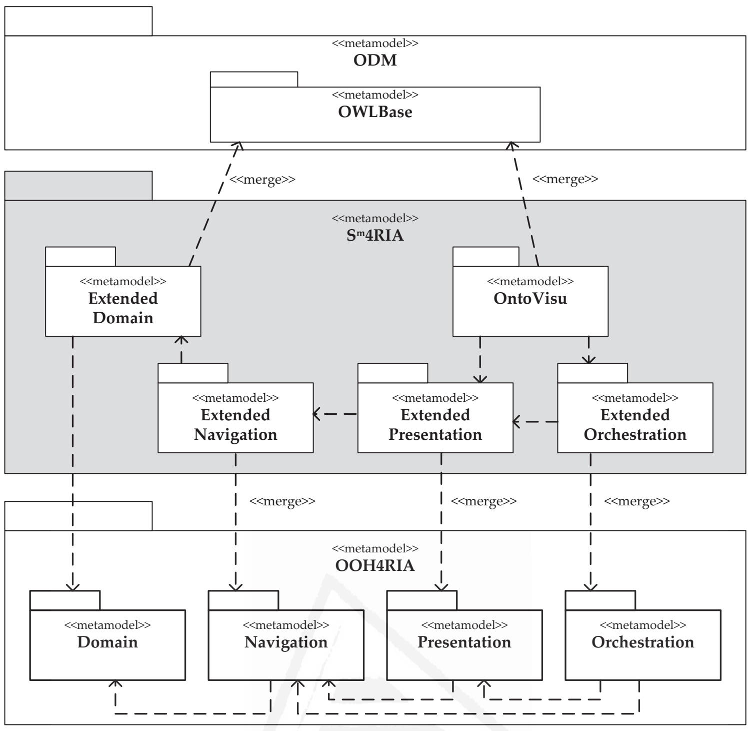


Figure 4.5. S^{m}4RIA meta-models and their links to the OOH4RIA and Ontology Definition meta-models.

The OOH4RIA metamodel is composed by four metamodels addressing different concerns: *Domain*, *Navigation*, *Presentation* and *Orchestration*. The S^{m}4RIA metamodel reuses the OOH4RIA Domain metamodel and extends the last three OOH4RIA ones. S^{m}4RIA defines two new metamodels, not included in OOH4RIA, i.e., the Extended Domain metamodel and the Ontovisu metamodel, which extend the ODM's *OWLBase* metamodel.

- *Extended Domain metamodel*. This metamodel extends the ODM OWLBase including elements not only for the design of ontologies ontologies but also the knowledge bases and external services (and their features: type, method of access, type of information) with the instances of the ontologies. The EDM is conformant to this metamodel.
- *Extended Navigation metamodel*. The Extended Navigation metamodel includes a new set of meta-elements that represent external navigational classes, attributes and operations, which are linked to elements of the EDM. The ENM is conformant to this metamodel.

- *Extended Presentation Metamodel*. The EPM extends the elements of the OOH4RIA Presentation Model adding new elements for the representation of static semantic annotations and the creation of UI contexts using external navigational classes from the ENM.
- *Extended Orchestration Metamodel*. The EOM adapts the OOH4RIA Orchestration model to the new EPM and ENM including or modifying the elements required to invoke external actions.
- *OntoVisu*. The last metamodel contains a set of elements that model the visualisation of the SRIA from the user's perspective. It allows describing the structural, behavioural and functional aspects of the UI elements in a platform-independent manner in contrast to the visual elements (widgets, panels) of the OOH4RIA Presentation and Orchestration metamodels, which depend on the SRIA implementation technology. In the same manner that the Extended Domain Metamodel, this metamodel extend the ODM OWLBase metamodel.

To sum up, the new S^{m}4RIA metamodels focus on three main tasks:

- To design the SRIA domain ontology and knowledge base – *Extended Domain Metamodel*. S^{m}4RIA allows defining the domain ontology of the application and the method of storage of the ontology instances, facilitating the creation of a knowledge base from the data stored in the SRIA database.
- To manage the access to ontologies and knowledge bases on the (Semantic) Web – Extended Domain and Navigation metamodels. S^{m}4RIA facilitates the reuse of existing ontologies in the SRIA domain ontology and includes primitives to access other ontology instances stored in other Web services.
- To manage the processes of semantic annotation and knowledge sharing within the SRIA – Extended Navigation Metamodel and OntoVisu. S^{m}4RIA metamodels defines conditions (based on the user data/preferences) to monitor the generation of domain ontology instances from the database tuples and how they are included within the SRIA interface.

4.2.5 Configurations of the S^{m}4RIA Development Process

As mentioned before, the main goal of the S^{m}4RIA methodology is to cover all the development phases of the SRIA proposal. This section shows how the S^{m}4RIA development process can be applied to two special cases of SRIA: the development of RI@BI applications (which contain an extended set of requirements, see Section 3.5), and the reengineering and modernisation of legacy applications into SRIAs, which is an interesting case given the high number of existing Web applications on the Web.

In order to address these development processes, two new subsets of components were included in S^{m}4RIA called S^{m}4RIA-B (Hermida et al., 2013), which groups the elements designed to address the development of RI@BI applications; and S^{m}4RIA-M, which contains the elements that support the reengineering and modernisation of legacy applications.

4.2.5.1 Addressing the Development of RI@BI Applications

The adoption of a new type of applications, such as RI@BIs, in business scenarios depends mainly on its maturity, benefits and development risks and costs. The application of model-driven methodologies to the development of RI@BIs can provide a solid framework for spreading the adoption of these applications. Model-driven engineering methodologies facilitate the representation of all the concerns of an application as software models, which can be subsequently employed to automatically generate the code of the application. In this manner, the cost of development and maintenance of the application can be minimised, thus encouraging the use of RIA platforms.

The S^{m}4RIA methodology includes a collection of modelling elements and processes specialised for the specification of BI SRIAs (labelled as S^{m}4RIA-B components). Given that this type of SRIAs needs to fulfil additional requirements, the S^{m}4RIA-B components are explained in a different section that the general design process for SRIA.

The following paragraphs describe the specific processes (or subtasks) included to the mentioned aim:

Activity 1. Design the SRIA (RI@BI) server (the affected tasks have been coloured in blue in the SPEM2 diagram depicted in Figure 4.6):

- During the definition of the EDM, designers can specify the external data structures required to invoke external services.
- During the definition of the ENM, designers can specify external service links to B2B services and can define mechanisms for mashing up knowledge from different sources from the Semantic Web. Furthermore, designers can indicate, for each navigational class, the navigational operations and the traversal links that will be offered as B2B services to other applications and their characteristics (e.g., type of service, access method, parameters).

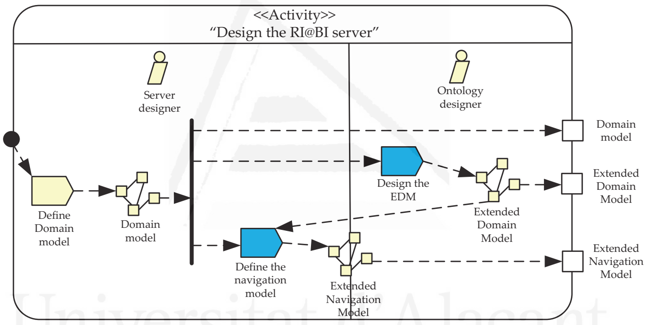


Figure 4.6. First activity of the S^{m}4RIA(-B) methodology.

Activity 2. Design the SRIA (RI@BI) client (the affected tasks have been coloured in blue in the SPEM2 diagram depicted in **Figure 4.7**):

- The Extended Presentation metamodel includes visualisation elements designed for BI SRIAs: new types of complex data analysis widgets, such as charts or maps.
- In the Extended Orchestration model, the presentation widgets can be linked to the mash-up methods previously defined in the ENM.

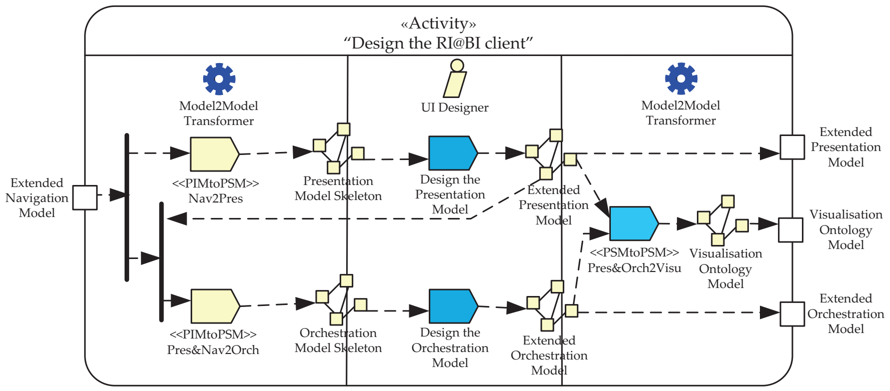

«Activity»
"Design the RI@BI client"
Model2Model Transformer
UI Designer
Model2Model Transformer
Extended Navigation Model
<<PIMtoPSM>> Nav2Pres
Presentation Model Skeleton
Design the Presentation Model
Extended Presentation Model
Extended Presentation Model
<<PSMtoPSM>> Pres&Orch2Visu
Visualisation Ontology Model
Visualisation Ontology Model
<<PIMtoPSM>> Pres&Nav2Orch
Orchestration Model Skeleton
Design the Orchestration Model
Extended Orchestration Model
Extended Orchestration Model

Figure 4.7. Second activity of the S^{m}4RIA(-B) methodology.

4.2.5.2 Addressing the Modernisation of Legacy Web Applications with S^{M}4RIA

During the past five years, the Web of Data has experienced a constant growth in the number of datasets freely available, supported and boosted from different organisations (with public or private funding) and users (from academia or industry). The existing dataset network is continuously enriched with data of any nature and topic from local databases, aiming at improving the permeability of the data in our societies. However, despite on-going efforts, a large amount of information flowing through the Web is still not available in these repositories, but stored in traditional Web applications or legacy Web applications. Unlike data repositories, Web applications are not static. The structure of the application can change during its development and exploitation, while the database can suffer major/minor modifications. Moreover, it is not unusual that the information about the design/development of a legacy Web application is not available.

In this context, the SRIA approach could improve the existing applications providing a rich user interface and a manner of sharing the knowledge/data contained as linked data. At the same time, the S^{m}4RIA methodology could support the process of transition between the legacy Web application and the resulting SRIA. This process of modernisation will be focused on the data of the Web application given that the main interest remains in sharing them as linked data.

In order to generate SRIAs from legacy applications, S^{m}4RIA includes a collection of modelling artifacts and processes (labelled as

S^m4RIA-M) that *1)* facilitate the extraction of the domain knowledge contained in the schema of the legacy database; *2)* specify the manner this knowledge will be mapped onto a Web resource; and *3)* generate the implementation of the software components of the new SRIA.

Unlike the development process of a new application from scratch, modernisation processes require a task(s) that extracts the information required for the models from a pre-existing source, which in this first case, is the structure of an existing database. In S^m4RIA, this task is included in the first activity of the development process, represented by an SPEM2 diagram depicted in Figure 4.8 (coloured in orange those tasks and models specific for modernisation processes).

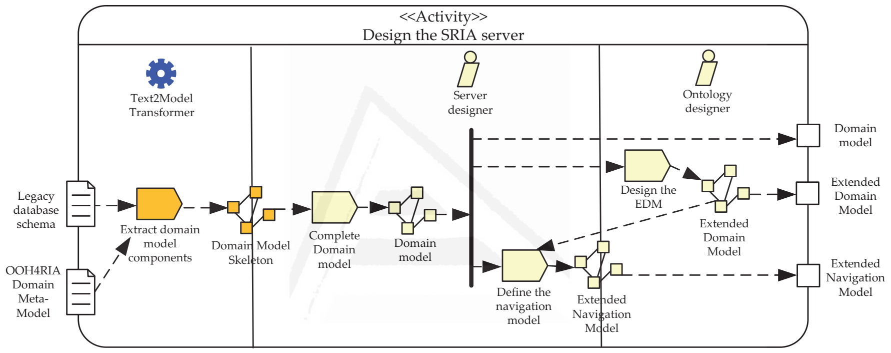


Figure 4.8. First activity of the S^m4RIA(-M) development process.

In this case, instead of specifying a new Domain model from the stakeholders' requirements, the first activity can start with a text-2-model transformation that extracts the main components of the database schema and creates a first version of Domain model. Subsequently, the server designer checks, and corrects if necessary, the model resulting from the transformation process. In addition, they need to include those elements that cannot be automatically obtained from the database: derivate attributes, class (custom) operations, compositions and aggregations.

Using the reengineered Domain model, in the next tasks, the designer builds the domain ontology of the application (at least one) using the EDM and links all its elements to the domain model previously obtained from the database. As mentioned before, both domain models are the basis for creating the rest of the S^m4RIA models.

This process of modernisation can be further automated using model-to-model transformations in some specific cases such as, the development of administrative views for legacy Web applications or (S)RIA. This task is usually performed in any project and the requirements and features of the resulting applications are normally similar independently from the application domain. Administrators mainly need a view which:

a) Can show any piece of data stored in the database;
b) Facilitates the invocation of the main CRUD operations and search operations over the domain objects;
c) Protected by an authentication process.

Given the simplicity of the features of this view, it is possible to automatically build most of the S^{m}4RIA design models that capture them. To this aim, in the first and second activities of the process, designers can apply a collection of model-to-model transformation processes as illustrated by the SPEM2 diagrams in Figure 4.9 and Figure 4.10 (the transformations are coloured in orange).

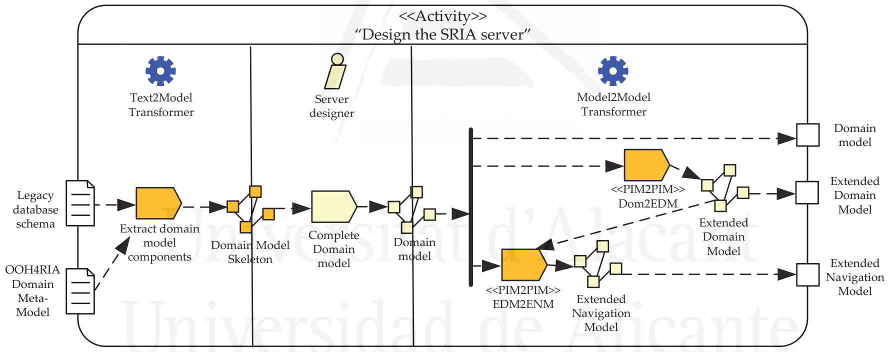


Figure 4.9. Modification of the first activity of S^{m}4RIA(-M) including M2M transformations.

In this way, once the back-end designer has completed the Domain model, the *Dom2EDM* and *EDM2ENM* M2M transformations can be invoked obtaining as a result the EDM and the ENM of the application, respectively. The first transformation obtains a domain ontology from the data entities defined in the Domain model. Moreover, the resulting EDM includes the database-ontology mapping rules and defines the basic query operations over ontology instances. Subsequently, the *EDM2ENM* transformation creates an ENM with all the possible navigational nodes and paths, and the invocations of the main CRUD operations for each data entity. In this process, it is supposed that there is only a single administrator in the Web application. If there were more than one with different security levels, it would be necessary to indicate which data objects can be accessed by each of them in the Domain model and, subsequently, generate a view of the ENM for each of them. Depending on the complexity and security needs, it might be more convenient to model this case manually.

In the second activity, the ENM is transformed into the EPM and EOM in the *ENM2EPM* and *ENM&EPM2EOM* M2M transformation processes. Both transformations build the structure and behaviour of the user interface based on a collection of navigational patterns. From each of the detected patterns, the first transformation creates a default set of components of the UI while the second one specifies the required ECA rules, which define the behaviour of the UI and the invocation of the SRIA server services.

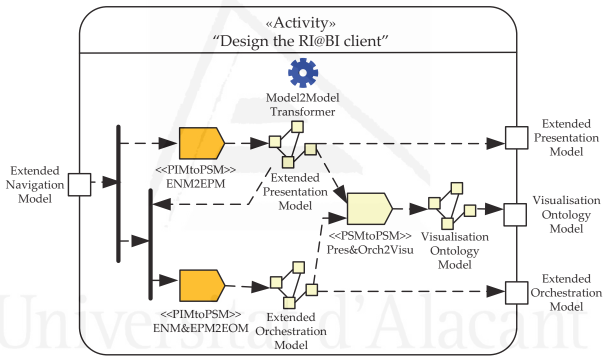

«Activity»
"Design the RI@BI client"
Model2Model Transformer
Extended Navigation Model
<<PIMtoPSM>> ENM2EPM
Extended Presentation Model
Extended Presentation Model
<<PSMtoPSM>> Pres&Orch2Visu
Visualisation Ontology Model
Visualisation Ontology Model
<<PIMtoPSM>> ENM&EPM2EOM
Extended Orchestration Model
Extended Orchestration Model

Figure 4.10. Modification of the second activity of S^{m}4RIA(-M) including M2M transformations.

It is worth noticing that although in this case most of the processes are performed automatically, it is also possible to include a task in which a designer checks the resulting models and adapt them to their needs. For instance, the style of the UI elements can be modified after the generation of the model even thought this can be also carried out once the application has been generated.

The automatic generation of the complete UI as shown in this process can be only performed with those Web applications in which the UI is usually very similar and its features do not vary depending on the application domain. In other cases, it is recommended to obtain a mock-up that can be completed by the UI designer or even create it from scratch.

Another example of this type of UI is the HTML views of raw Linked data such as those generated by the some of the main Linked Data managers, e.g., OpenLink Virtuoso or D2RQ. These views are normally used for the visualisation of the RDF content of the repository with a user-friendly HTML view, which could be also indexed by the Web search engines. In order to generate (S)RIA rich user interfaces for the Linked Data repositories, which improve the interaction of traditional Web interfaces with the human users, it would be also possible to use an alternative approach that takes as input the domain ontology (-ies) or vocabulary (-ies) on which the Linked Data repository is based on.

With this aim, the first activity of the S^{m}4RIA development process can be reconfigured as depicted in Figure 4.11. In this last version of the first activity, from the domain ontology (-ies) of the repository, a text-2-model process extracts the main elements of the ontology and creates a mock-up of the EDM, which can be completed or modified by the ontology designer. Once completed, the *EDM2Domain* and *EDM2ENM* transformation processes build the Domain model and the ENM from the EDM. The *EDM2Domain* transformation creates a domain model with the main data structures for the management and local storage (if necessary) of the ontology instances.

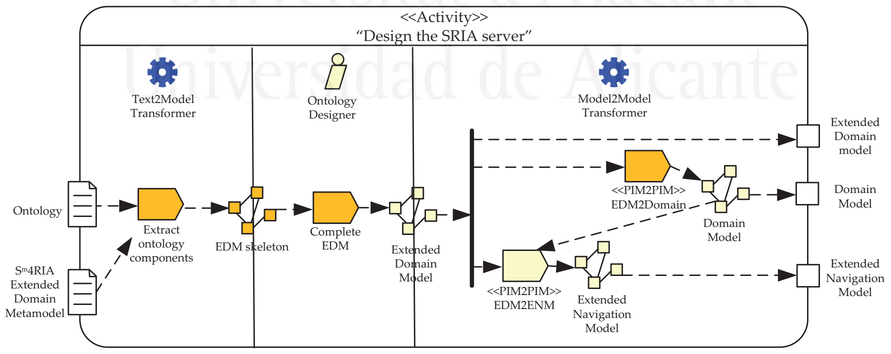


Figure 4.11. Modification of the first activity of S^{m}4RIA(-M) for the generation of RIA interfaces for Linked Data repositories.

The "*Extract ontology components*" task performs different subtasks depending on the representation language of the input ontology. It should be taken into consideration that the goal of the EDM metamodel is the representation of lightweight ontologies and it cannot therefore represent complex ontology elements such as user axioms or rules.

4.3 Conclusions

This chapter has introduced the main contributions of this thesis: the SRIA approach and the S^{m}4RIA methodology.

The content of RIAs cannot be easily crawled and indexed by Web search engines, which can prevent developers from implementing RIA applications and enterprises from using this type of application for their business. In this context, the first contribution is **the definition of a new class of RIA, i.e. the *Semantic* RIA**, whose main goal is to provide a generic solution to this problem based on the existing Semantic Web techniques and independent from the implementation technology. The SRIA approach extends the structure of the traditional RIA including modules for sharing local knowledge and consuming knowledge from the Semantic Web. The knowledge managed by the SRIA is completely available as linked data using the annotation model proposed. This three-layer model combines different existing approaches and can provide a complete view of the application from the user perspective, hiding design or implementation details, in such a way that this knowledge could be reused to create widgets for other Web or desktop applications.

The three case studies developed have been used to assess the proposal in three different scenarios: the Semantic Web with the development of a media player, the Social Semantic Web with the development of a SNS, and the field of Business Intelligence for enterprises. The qualitative assessment was performed either internally, i.e., analysing the features developed and the possible improvements, and externally, i.e., in international conferences and journals. The evaluation of performance and scalability issues was not in the scope of the analysis carried out.

The main conclusion obtained from the internal analysis was that the complexity of the resulting SRIA architecture notably increased the cost of development and maintenance (comparing to RIA or traditional Web applications), which is a risk factor that threatens the viability of this type of applications and their future success, especially when working in business scenarios. SRIAs require complex software modules combined in a single architecture in order to deal with issues of knowledge management and visualisation.

Model-driven Web engineering methodologies facilitate the development and maintenance of complex Web applications such as SRIAs. As an example of such a methodology, the thesis introduces the second contribution, i.e., **the S^{m}4RIA methodology**, which offers a set of models that can represent the structure and behaviour of their software components, and the processes that transform these models into the final code. Thus, S^{m}4RIA, as a model-driven methodology, can minimise the identified risks and can bring many benefits to the development of complex RIA (in the Social Web, for Business Intelligence or in modernisation processes). As shown in this chapter, the proposal is sufficiently flexible to cover the different case studies, each with their own requirements.

The following chapters will explain the three activities of the S^{m}4RIA process and its variations in depth using the SNS case study as example. Each chapter will describe the tasks of the activity, the S^{m}4RIA metamodels and models involved, the transformation processes (model-to-model, model-to-text, or text-to-model) and the different decisions taken during the design of S^{m}4RIA. Further on, they will explain the possible S^{m}4RIA extensions or variations presented in Section 4.2.5.

Chapter 5. Designing the Server Components of a Semantic Rich Internet Application

In the development of applications with a client-server architecture, such as Web applications, the application server is usually the first element that is developed. The server components normally store and manage the application data that is employed by the client components, which directly interact with the final users. Following the same motivation, the first activity of the S^m4RIA development process is the development of the SRIA server.

Specifically, this activity is composed of three tasks performed by two actors (please see the SPEM2 diagram in Figure 4.2, page 70), i.e., the server designer and the ontology designer. The output artefacts of the first activity are the first three S^m4RIA models: the *Domain Model*, the *Extended Domain Model* and the *Extended Navigational Model*. The main aims of this activity are the following:

a) To design the data structures and the operations that can be performed over them.
b) To build the domain ontology of the application, importing external ontologies, if necessary.
c) To specify the main operations over ontology instances.
d) To specify the external sources of Linked Data that can be used by the SRIA.

e) To design the main navigational paths through the application data that can be followed using the SRIA client, thus specifying the service interface that will provide data to the SRIA client(s).
f) To design the object-relational and ontology-relational mappings.
g) To specify the service interface that will provide data to the SRIA client and ontology instances to external clients.

This chapter describes in detail the three tasks of the first activity in separate sections using the case study explained in Section 3.4.1 (page 54): the development of a social network site. The following sections will explain the process followed in each of the tasks, including a detailed description of the models and metamodels employed. The models resulting from the design of the rest of the case studies proposed are included in Annex D (page 267).

5.1 Design the Strategies of Data Persistence

In the first task, from a list of requirements from the stakeholders, the server designer defines a) the main data structures using an Object-oriented paradigm, which will be managed by the application and stored in the database; and b) the operations over these structures, which will be invoked by other server or client components.

As a result of this process, the designer creates the Domain model of the application. This model is conformant to the OOH4RIA Domain metamodel, which extends the UML metamodel for class diagrams with new modelling elements for the definition of collection types, operation types and object-relational mappings. The decisions taken during the design of the Domain model will strongly influence the rest of the process, since it is, directly or indirectly, the input of all the tasks of the process. This model does not contain technical or implementation details.

5.1.1 The Domain Metamodel

The Domain metamodel extends the UML metamodel for class diagrams (abstract and concrete syntaxes) introducing several modifications in order to remove ambiguous aspects in the creation of

Object-Relational mappings (mapping rules that transform the data structures into registers of a relational database and vice versa):

(i) this metamodel defines a topology of operation types, such as *create*, *delete*, *relate*, *unrelate*, *modify*, *readAll*, etc., each of which is related to a specific function in the final implementation (e.g., to create an object, to delete an object, etc.);

(ii) it represents the main collection types, e.g., *set*, *bag*, *list*, etc.;

(iii) the metamodel defines the concept of *object identifier* as a metaproperty of those class attributes that will be generated as primary keys, and the metaproperty "database alias" within classes, attributes and association roles for naming tables, columns and foreign keys, respectively.

Figure 5.1 depicts an UML class diagram with the abstract syntax of the Domain metamodel. The concrete syntax of the Domain metamodel was reused from the UML class metamodel. The abstract syntax shown in this figure was created using the EMOF metamodel, which facilitates the implementation of the model in a CASE tool. This metamodel is similar to the domain models of other model-driven methodologies such as OO-H, UWE or WebSA.

The main metaclasses of the Domain metamodel can be described as follows:

- **ConceptualModel**. This metaclass represents a complete model or a package (submodel) within the main model. A conceptual model contains a set of *ConceptualElement* elements even new *ConceptualModel* elements. It is illustrated using the concrete syntax of UML Package.
- **ConceptualElement**. This abstract metaclass represents any possible element contained in a Domain model (or *ConceptualModel* element).
- **Class**. This metaclass represents a type of data object characterised by its name. Similarly to the UML metamodel, a class contains a set of attributes and operations. It is depicted using the concrete syntax of the UML Class.
- **Attribute**. This metaclass represents a feature of a class or an association role, in both cases, characterised by its name and its primitive type (and collection type). It is depicted using the concrete syntax of the UML Attribute or the association

role, depending on which object is linked to. In this metamodel, the visibility of the class attribute (public, private or protected) is represented using a coloured circle on the left side of the attribute name. Those class attributes that are object identifiers (OID), whose *isOID* property is set *true*, are represented with the icon of a key.

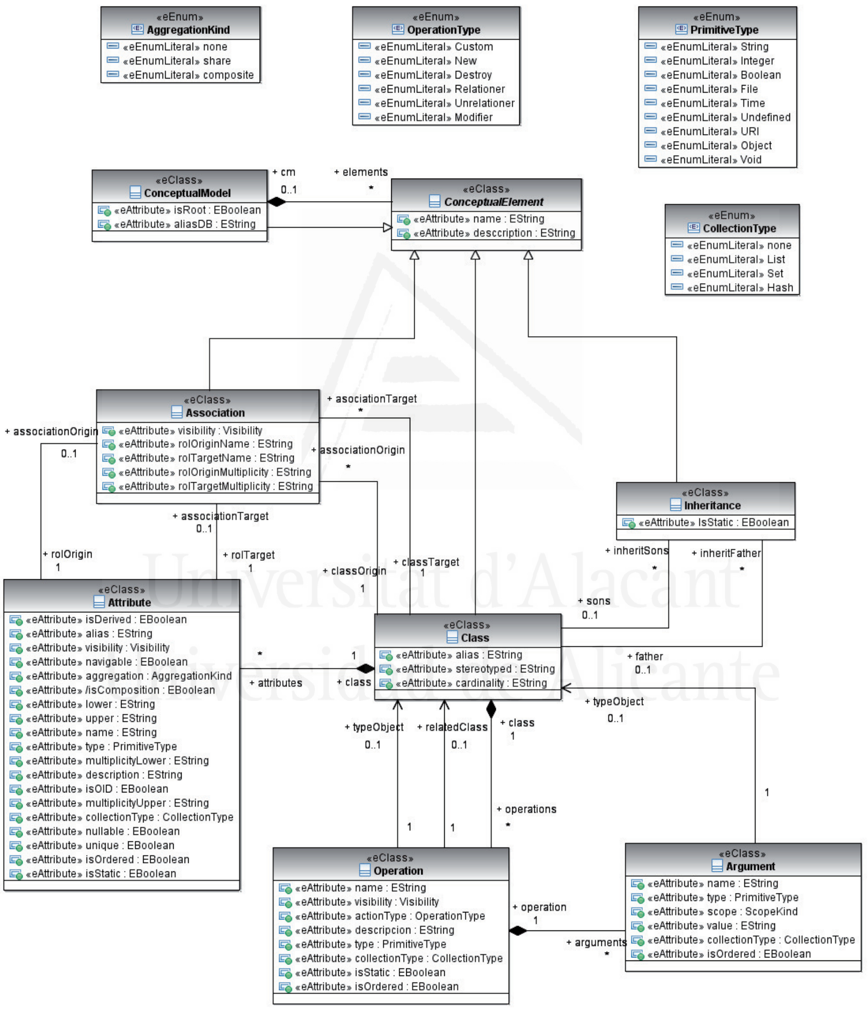

«eEnum» AggregationKind
«eEnumLiteral» none
«eEnumLiteral» share
«eEnumLiteral» composite
«eEnum» OperationType
«eEnumLiteral» Custom
«eEnumLiteral» New
«eEnumLiteral» Destroy
«eEnumLiteral» Relationer
«eEnumLiteral» Unrelationer
«eEnumLiteral» Modifier
«eEnum» PrimitiveType
«eEnumLiteral» String
«eEnumLiteral» Integer
«eEnumLiteral» Boolean
«eEnumLiteral» File
«eEnumLiteral» Time
«eEnumLiteral» Undefined
«eEnumLiteral» URI
«eEnumLiteral» Object
«eEnumLiteral» Void
«eClass» ConceptualModel
«eAttribute» isRoot : EBoolean
«eAttribute» aliasDB : EString
+ cm
0..1
+ elements
*
«eClass» ConceptualElement
«eAttribute» name : EString
«eAttribute» desccription : EString
«eEnum» CollectionType
«eEnumLiteral» none
«eEnumLiteral» List
«eEnumLiteral» Set
«eEnumLiteral» Hash
«eClass» Association
«eAttribute» visibility : Visibility
«eAttribute» rolOriginName : EString
«eAttribute» rolTargetName : EString
«eAttribute» rolOriginMultiplicity : EString
«eAttribute» rolTargetMultiplicity : EString
+ asociationTarget
*
+ associationOrigin
0..1
+ associationOrigin
*
+ associationTarget
0..1
+ rolOrigin
1
+ rolTarget
1
+ classOrigin
1
+ classTarget
1
«eClass» Inheritance
«eAttribute» IsStatic : EBoolean
+ inheritSons
*
+ inheritFather
*
+ sons
0..1
+ father
0..1
«eClass» Attribute
«eAttribute» isDerived : EBoolean
«eAttribute» alias : EString
«eAttribute» visibility : Visibility
«eAttribute» navigable : EBoolean
«eAttribute» aggregation : AggregationKind
«eAttribute» /isComposition : EBoolean
«eAttribute» lower : EString
«eAttribute» upper : EString
«eAttribute» name : EString
«eAttribute» type : PrimitiveType
«eAttribute» multiplicityLower : EString
«eAttribute» description : EString
«eAttribute» isOID : EBoolean
«eAttribute» multiplicityUpper : EString
«eAttribute» collectionType : CollectionType
«eAttribute» nullable : EBoolean
«eAttribute» unique : EBoolean
«eAttribute» isOrdered : EBoolean
«eAttribute» isStatic : EBoolean
*
+ attributes
1
+ class
«eClass» Class
«eAttribute» alias : EString
«eAttribute» stereotyped : EString
«eAttribute» cardinality : EString
+ typeObject
0..1
+ typeObject
0..1
+ relatedClass
0..1
+ class
1
+ operations
*
1
1
1
«eClass» Operation
«eAttribute» name : EString
«eAttribute» visibility : Visibility
«eAttribute» actionType : OperationType
«eAttribute» descripcion : EString
«eAttribute» type : PrimitiveType
«eAttribute» collectionType : CollectionType
«eAttribute» isStatic : EBoolean
«eAttribute» isOrdered : EBoolean
+ operation
1
+ arguments
*
«eClass» Argument
«eAttribute» name : EString
«eAttribute» type : PrimitiveType
«eAttribute» scope : ScopeKind
«eAttribute» value : EString
«eAttribute» collectionType : CollectionType
«eAttribute» isOrdered : EBoolean

Figure 5.1. Schema of the Domain EMOF Metamodel.

- **Operation**. This metaclass represents an action that can be performed over the class objects. Each operation is identified by its name the return type and the number, names and types of the arguments it contains (*Argument* metaclass). In this metamodel, each operation has an operation type (*OperationType* enumeration) that indicates if it is a CRUD operation or a custom operation. It is represented using the concrete syntax of the UML operations (including the modification of the representation of the *visibility* property).

- **Association**. This metaclass captures the semantics of a relationship between two classes in the same manner as the UML metamodel. It contains two attributes that establish two different relationships the direct one (from class A to B) and the inverse one (from B to A). It is depicted using the concrete syntax of the UML Association.
- **Inheritance**. This metaclass captures and represents the semantics of the UML class inheritance using the same concrete syntax. Multiple inheritance is not supported by this metamodel.

The following section presents the Domain model of the SNS case study as an example. This example contains all the elements of the Domain metamodel described in this section.

5.1.2 An Example of Domain Model: Social Network Site

This section commences the explanation of the manner in which S^{m}4RIA should be applied to develop a SRIA application with a specific case study: the Social Network site. It will deeply describe the functionalities and components of the case study, including some aspects not explicitly mentioned in the initial description (see Section 3.4.1, page 54).

In this case, after obtaining all the requirements from the stakeholders, the server designer(s) created the Domain model illustrated in Figure 5.2. In this model, there are two types of possible members: community members (or simply, users, class *UserAccount*) and corporative members (artists, i.e. users with a profile of the class *Artist*). The first ones are the main contributors of the SNS, i.e., those who interact, create and consume the contents of the SNS. Corporative users also have the possibility to manage an official profile to interact with their fans or followers. This profile can only be created by the administrator of the application.

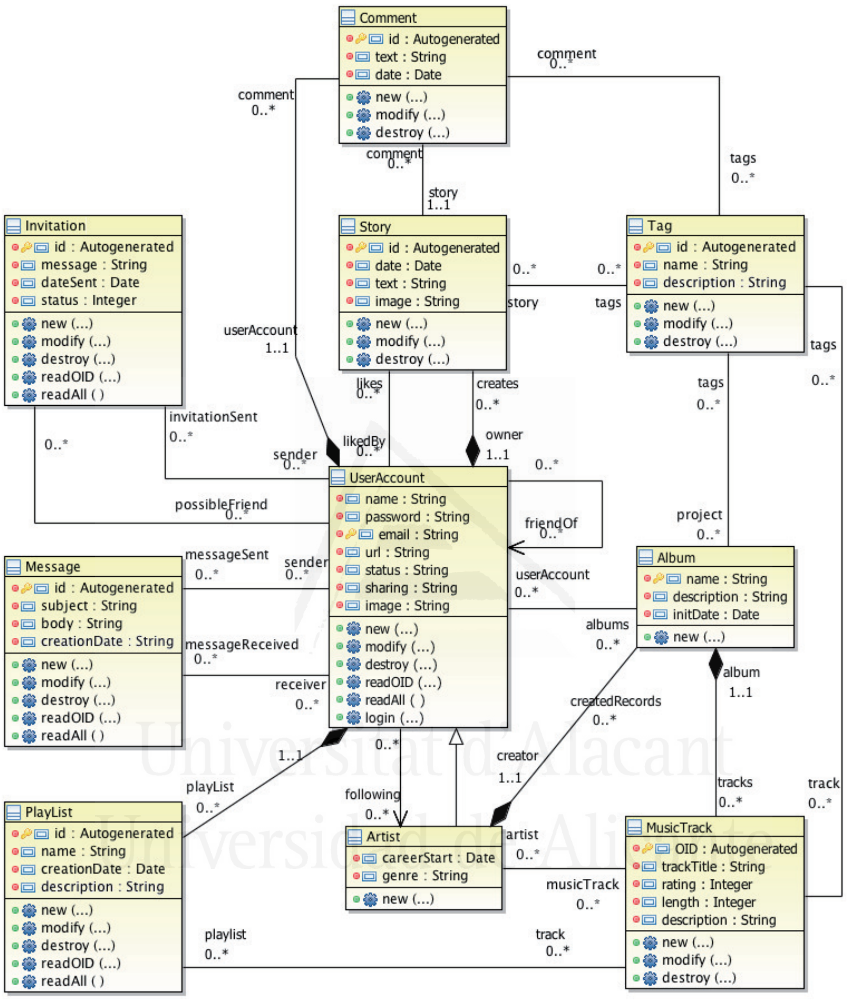


Figure 5.2. Diagram of the Domain Model of the Social Network Site case study.

Users can be connected by a relationship of friendship, which is established by a mechanism based on invitations (class *Invitation*). All the members own a wall where they can publish their own stories (class *Story*) and can read their friends' stories. Moreover, they can write comments about their stories, their friends' stories or replying other comment (class *Comment*). Users can also send private messages (class *Message*) to their friends.

As this SNS is focused on the music domain, artists can publish their own albums (class *Album*), containing a list of tracks (class *MusicTrack*).

User can create his/her own personal playlists from the albums published by the artists. Albums, music tracks, stories and comments can be tagged by any user (class *Tag*) in such a way that the tags will facilitate the retrieval of elements with similar content.

As can be appreciated from the diagram, each class includes a collection of attributes that characterise them and also defines the main CRUD operations (*new*, *modify*, *destroy*, *readAll* and *readOID*). It is also worth highlighting that the *sharing* attribute (*UserAccount* classes) will allow the application to store the user's privacy preferences regarding the exportation of the data as ontology instances in further stages of the S^{m}4RIA process.

5.2 BUILD THE DOMAIN ONTOLOGY AND THE ONTOLOGY-DATABASE MAPPING RULES

Once the Domain model is completed, the ontology designer addresses the development of the domain ontology of the application with the Extended Domain Model, which captures and represents the domain knowledge as a lightweight ontology and specifies the reuse of external ontologies and knowledge bases within the SRIA. Thus, this model could be partially considered as an ontology representation language.

The model, based on the Extended Domain metamodel, is aimed at linking the domain knowledge with other sources of knowledge. The S^{m}4RIA methodology does not propose a new ontology design methodology but leaves this decision to the ontology designer, who can use the one that better fits with their background and the model.

The EDM can be built in two stages that can be carried out iteratively. In the first one, the designer defines the local ontology and establishes a relationship between the local ontology and the external ontologies that will be imported (if necessary). Furthermore, they define the manner in which the instances of each ontology (local or external) are stored, i.e., they define the available knowledge bases and the manner of access to them (type of service, URI, etc.)

Designers can import an ontology for three main reasons: *a)* because they need to reuse some of the elements (concepts or properties), thus reducing the effort of building a new ontology; *b)* because they need to export the information contained in the SRIA as instances of the imported ontologies or *c)* because they need to access to new information stored in an external repository as linked data. In any case, the ontology designers need to link the external elements to elements of the local ontology. The elements of the imported ontologies cannot be modified, just extended, since the definition is managed by other organisations. Despite this, it is possible to link elements of imported ontologies to elements of the Domain Model of the application.

In a second stage, the designer builds the actual domain ontology creating new elements or reusing elements from the ontologies already imported (concepts from each source can be aligned using different mechanisms, e.g., inheritance or equivalency, among others).

5.2.1 THE EXTENDED DOMAIN METAMODEL: ABSTRACT AND CONCRETE SYNTAXES

The Extended Domain metamodel, whose abstract syntax is depicted in the diagram of Figure 5.3, represents the elements needed for the representation of lightweight ontologies and mapping the elements of the ontology to the elements of the Domain Model.

This metamodel extends the ODM into three directions:

(i) it defines the concept of "*knowledge source*" in order to allow the specification of the repositories of ontology instances (or knowledge bases) that are related to any ontology;

(ii) the metamodel includes the elements required to define relations between the ontology elements and the elements from the Domain Model, thus manually specifying the ontology-database mapping rules;

(iii) It defines a collection of operations that can be performed over the ontology instances (mainly query operations such as *readAll*, *filter*, etc.)

The diagram depicted in Figure 5.3 shows the main elements of the Extended Domain metamodel and the connections to the elements of the Domain metamodel (coloured in blue) and *OWLBase* metamodel (coloured in red). Table 5.1 describes each of the elements of the ED metamodel contained in the diagram and presents their concrete syntax (or graphical notation).

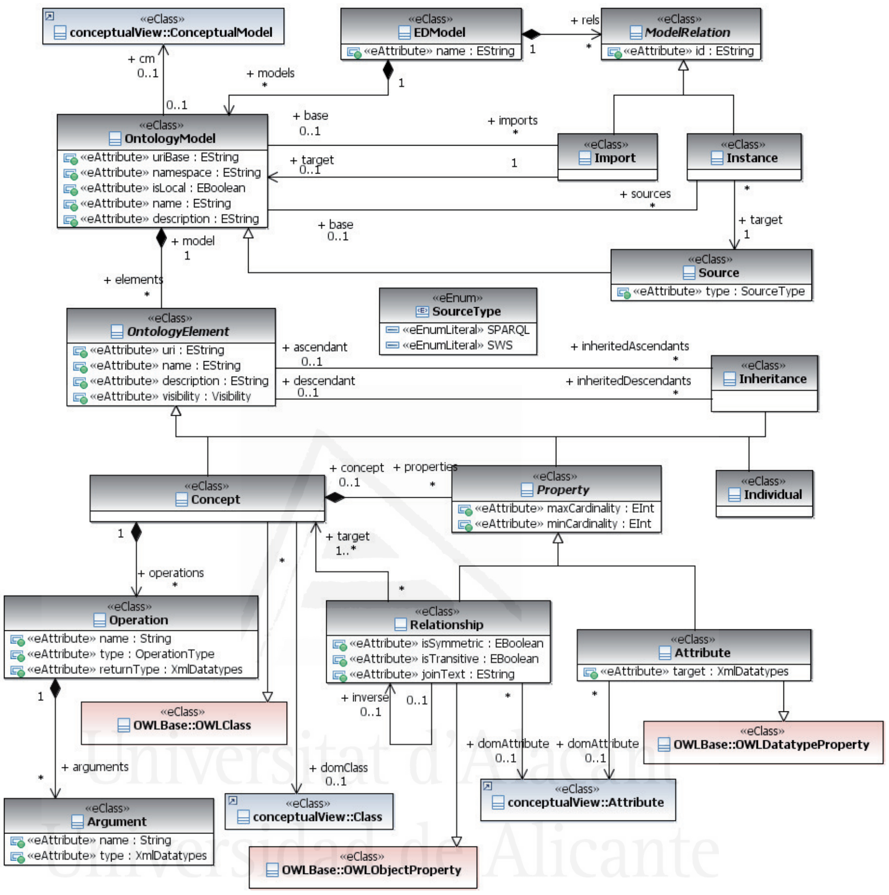
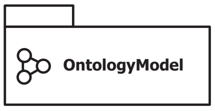
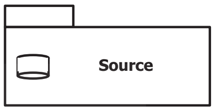
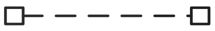
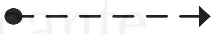
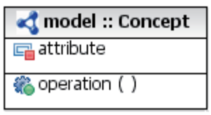
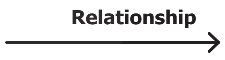
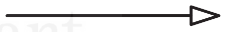

«eClass» conceptualView::ConceptualModel
«eClass» EDModel
«eAttribute» name : EString
+ rels
1
*
«eClass» ModelRelation
«eAttribute» id : EString
+ cm
0..1
+ models
*
1
0..1
«eClass» OntologyModel
«eAttribute» uriBase : EString
«eAttribute» namespace : EString
«eAttribute» isLocal : EBoolean
«eAttribute» name : EString
«eAttribute» description : EString
+ base
0..1
+ imports
*
+ target
0..1
1
«eClass» Import
«eClass» Instance
+ sources
*
+ target
1
+ base
0..1
+ model
1
+ elements
*
«eClass» Source
«eAttribute» type : SourceType
«eEnum» SourceType
«eEnumLiteral» SPARQL
«eEnumLiteral» SWS
«eClass» OntologyElement
«eAttribute» uri : EString
«eAttribute» name : EString
«eAttribute» description : EString
«eAttribute» visibility : Visibility
+ ascendant
0..1
+ descendant
0..1
+ inheritedAscendants
*
+ inheritedDescendants
*
«eClass» Inheritance
«eClass» Concept
+ concept
0..1
+ properties
*
«eClass» Property
«eAttribute» maxCardinality : EInt
«eAttribute» minCardinality : EInt
«eClass» Individual
1
+ operations
*
+ target
1..*
«eClass» Operation
«eAttribute» name : String
«eAttribute» type : OperationType
«eAttribute» returnType : XmlDatatypes
«eClass» Relationship
«eAttribute» isSymmetric : EBoolean
«eAttribute» isTransitive : EBoolean
«eAttribute» joinText : EString
«eClass» Attribute
«eAttribute» target : XmlDatatypes
«eClass» OWLBase::OWLClass
+ inverse
0..1
0..1
«eClass» OWLBase::OWLDatatypeProperty
+ domAttribute
0..1
+ domAttribute
0..1
+ arguments
*
+ domClass
0..1
«eClass» conceptualView::Attribute
«eClass» Argument
«eAttribute» name : String
«eAttribute» type : XmlDatatypes
«eClass» conceptualView::Class
«eClass» OWLBase::OWLObjectProperty

Figure 5.3. Diagram of the main elements of the Extended Domain metamodel.

Table 5.1. Description of the main elements of the Extended Domain metamodel.

Metaelement	Description	Graphical Notation
EDModel	This metaclass represents the whole model and can contain elements of the *OntologyModel* and *ModelRelation* metaclasses.	*No graphical representation.*
OntologyModel	This metaclass extends the *OWLBase::OWLOntology* metaclass and represents an ontology. *OntologyModel* elements contain a collection of	

Metaelement	Description	Graphical Notation
	OntologyElement elements and can be related to a list of *Source* elements, i.e., knowledge bases, by means of the *Instance* metaelement. An *OntologyModel* element can be linked to one *conceptualView::ConceptualModel* element (from the *Domain* model) at most.	
Source	This metaclass represents a knowledge base (generally speaking, a "knowledge source"), i.e. a collection of ontology instances. A *Source* element must be related to at least one *OntologyModel* element, in the same way that ontology instances must be related to their ontology classes.	
Instance	Association between a *Source* element with its corresponding *OntologyModel* elements. The Individual elements of a *OntologyModel* element can be stored in several Source elements.	
Import	Association between two *OntologyModel* elements that represents the mechanism for ontology import. A DOM can import elements from other DOMs.	
OntologyElement	Abstract metaclass that represents all the types of elements that can be included in a knowledge definition.	*No graphical representation.*
Concept	This metaclass extends the *OWLBase::OWLClass* and represents a concept of an ontology. Each *Concept* element can be related to a single Domain Class element (or *conceptualView::Class*). The graphical notation is similar to the Domain Class notation. It is depicted as a box with three compartments: concept information, attributes and operations.	

Metaelement	Description	Graphical Notation
Property	Abstract metaclass that represents all the possible features of a Concept element.	*No graphical representation.*
Attribute	This metaclass represents a feature of a concept. It extends the *OWLBase::OWLDatatypeProperty*. Each *Attribute* element can be related to at least a Domain Attribute element (or *conceptualView::Attribute*).	*See the Concept metaelement.*
Relationship	This metaclass extends the *OWLBase::OWLObjectProperty* and represents a relationship between two Concept elements. Each *Relationship* element can be related to at least a *conceptualView::Attribute* element (from the Domain model). The end of the arrow points to the	
Inheritance	This metaclass represents a special type of relationship between Concept elements or between Property elements. Relationship elements connect elements related by a generalisation/specialisation relationship	
Operation	This metaclass represents an action that can be performed over ontology instances.	*See the Concept metaelement.*

As can be noticed, the concrete syntax of the Extended Domain model is similar to the notation of the Domain model in order to facilitate the process of modelling to the back-end designers.

Finally, Table 5.2 introduces a collection of the main OCL constrains defined over the Extended Domain metamodel, which aim at preserving the consistency of the information stored in the model.

Table 5.2. Summary of the OCL constraints for the Extended Domain metamodel.

Name	Description / OCL Statement
DifferentModelUris	Two *OntologyModel* elements cannot refer to the same URI.

	`context EDModel`
	`inv:`
	`self.models->forEach(m | self.models.uriBase->select(n | n = m.uriBase)->size() = 1)`

LocalModelWithCM	Local *OntologyModel* elements must be associated to a *ConceptualModel* element.

	`context OntologyModel`
	`inv:`
	`!self.isLocal || (self.isLocal && self.cm <> null)`

ImportModel	The model imported and the base model must be different.

	`context Import`
	`inv:`
	`self.base <> self.target`

AllPropertiesDifferent	All the properties of a *Concept* element must be different.

	`context Concept`
	`inv:`
	`self.properties->forAll(p | self.properties->select(p2 | p.uri = p2.uri && p.name = p2.name)->size() = 1)`

InverseRelationship	A *Relationship* element cannot be its own inverse.

	`context Relationship`
	`inv:`
	`self.inverse <> self`

Generalisation	Ascendant and descendant *Concept* elements must be different.

	`context Inheritance`
	`inv:`
	`self.ascendant <> self.descendant`

The next section presents an example of Extended Domain Model following the development of the SNS case study.

5.2.2 The Extended Domain Model for the Social Network Site

The ontology designer continues the design of the SNS case study by creating the Extended Domain Model. As mentioned before, this model can be built in two stages. In the first one, the designer defines the local ontology, the ontologies that are imported and the knowledge sources available, which contain the ontology instances of the external ontologies. Figure 5.4 illustrates the Package view of the EDM of the SNS resulting from the first stage of this task.

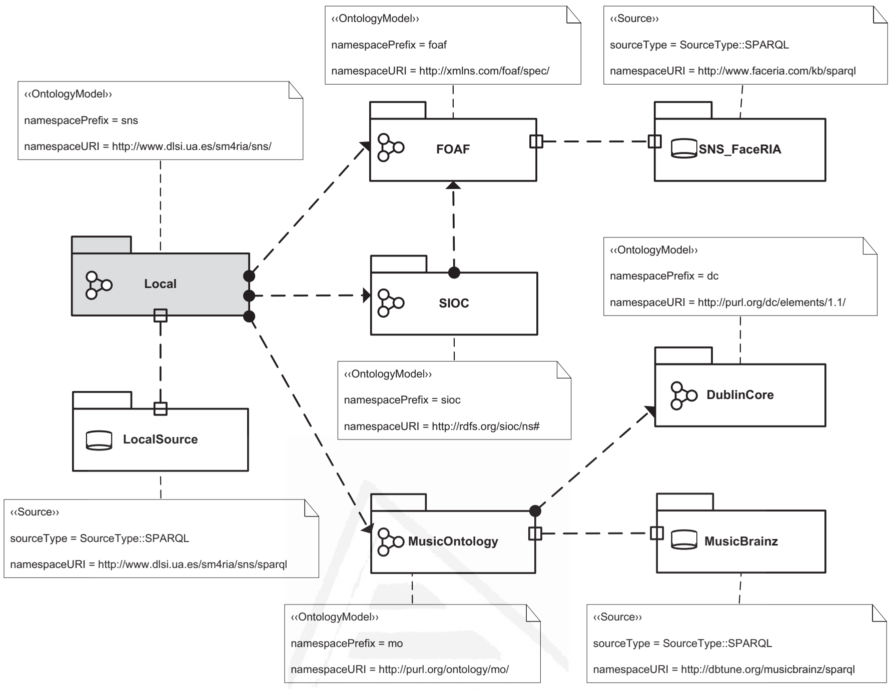

‹‹OntologyModel››
namespacePrefix = foaf
namespaceURI = http://xmlns.com/foaf/spec/
‹‹Source››
sourceType = SourceType::SPARQL
namespaceURI = http://www.faceria.com/kb/sparql
‹‹OntologyModel››
namespacePrefix = sns
namespaceURI = http://www.dlsi.ua.es/sm4ria/sns/
FOAF
SNS_FaceRIA
Local
SIOC
‹‹OntologyModel››
namespacePrefix = dc
namespaceURI = http://purl.org/dc/elements/1.1/
‹‹OntologyModel››
namespacePrefix = sioc
namespaceURI = http://rdfs.org/sioc/ns#
DublinCore
LocalSource
‹‹Source››
sourceType = SourceType::SPARQL
namespaceURI = http://www.dlsi.ua.es/sm4ria/sns/sparql
MusicOntology
MusicBrainz
‹‹OntologyModel››
namespacePrefix = mo
namespaceURI = http://purl.org/ontology/mo/
‹‹Source››
sourceType = SourceType::SPARQL
namespaceURI = http://dbtune.org/musicbrainz/sparql

Figure 5.4. Extended Domain Model of the SNS case study (Package view).

The *Local* OntologyModel element represents the domain ontology of the application, while the *LocalSource* Source element indicates where their instances are stored in. Both elements must be included in any EDM by default. The properties of each element are included within the annotation elements attached to them. In this example, the *Local* OntologyModel element imports three ontologies:

- the FOAF ontology (OntologyModel *FOAF*), which represents people, documents and their relationships;
- SIOC (OntologyModel *SIOC*), which represents concepts related to social sites and the Social Web;
- the MusicOntology ontology (OntologyModel *MusicOntology*), which represents concepts about the music domain; and
- the Dublin Core ontology (OntologyModel *DublinCore*), which is not directly imported but is required by the *MusicOntology* OntologyModel element in order to represent some of the properties of their concepts. The ontology designer should also

analyse and choose which ontologies are also needed by the ones imported.

The designed SNS aims to reuse the information about the users of another existing SNS called *FaceRIA* using its SPARQL service in order to facilitate the interconnection of both SNS. Furthermore, the stakeholders need to retrieve information from the MusicBrainz repository. As a consequence, the designer included the *MusicBrainz* Source element, which specifies the service that provides access to the *MusicBrainz* ontology instances, represented by the MusicOntology. By importing the MusicOntology ontology, the designer aims at reusing the elements for the local ontology, publishing the application data as instances of this ontology (in this way, the information could be reused by MusicBrainz for instance) and accessing to the information stored in MusicBrainz.

Once this first stage is completed, the designer builds the domain ontology (i.e., specifies the content of the *Local* OntologyModel element), creating new ontology elements, reusing elements from the ontologies already imported or mapping local concepts to the concepts from external ontologies. Figure 5.5 depicts a fragment of the final domain ontology for the SNS case study. The names of the elements imported from other ontologies include a prefix corresponding with the namespace of its OntologyModel element. For instance, prefixes "sioc::" (for concepts), "sioc__" (for properties) and "foaf__" correspond to the namespaces of the SIOC and FOAF OntologyModel elements respectively, defined in the first stage.

Following the example, the elements of the domain ontology can be classified into two groups (in the same way that the Domain model elements):

- a group with the elements that represent the social entities (with the classes *sioc__UserAccount*, *foaf__Person*, *Story*, *Comment*, *Tag*; and their properties), which would be similar for any SNS and contains those elements required for the representation of the social interaction of a community of users; and
- another group dependent from the application domain, which is, in this case, associated to the music domain (classes *mo::MusicArtist*, *mo::Track*, *mo::Record* and *PlayList*) and determines the main aim of the application.

In this example, the designer reused elements from FOAF and SIOC in order to represent the users of the SNS (concept *foaf::Person*) and their accounts within the application (concept *sioc::UserAccount*), which can facilitate the interchange of the knowledge contained in this SNS with other social sites. Furthermore, they employed elements of MusicOntology and DublinCore for the representation of the artists (concept *mo::Artist*), albums (concept *mo::Record*) and tracks (concept *mo::Track*) managed by the application. The concepts that represent the stories (concept Story), comments (concept *Comment*), playlists (concept *PlayList*) and tags (concept *Tag*) were created by the designer within the context of the *Local* OntologyModel element.

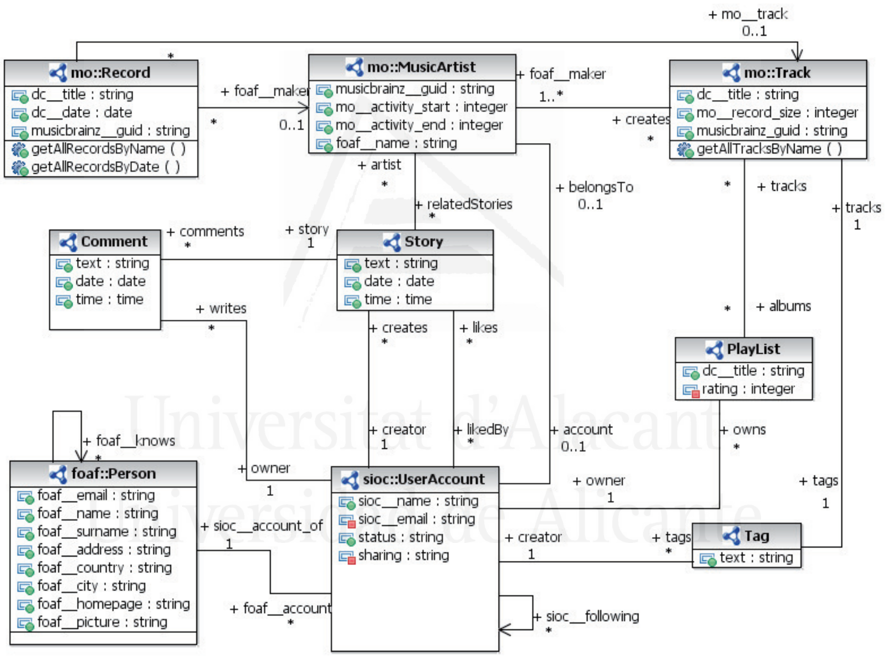


Figure 5.5. Diagram of the content of *Local* Ontology Model element for the SNS case study.

This model does not contain the concepts of message or invitation that were included in the Domain model. Not all the concepts captured by the Domain model should be included in the EDM. The elements of this last model might be linked to elements of the Domain model but, depending on the requirements, the ontology designer could also use new concepts from an external source, which might not be associated to any element of the Domain model. The link between the elements of the two diagrams (at any level: concept or property) creates a mapping rule that could be used in order to generate the information contained in the

SRIA as instances of a certain ontology or to manage external information as if it were local, i.e., mapping the external data structures into the local representation.

The attributes of each concept are represented within its compartment. For each attribute, the diagram depicts its name, type (XML data type) and visibility. The semantics of the visibility in the EDM differs from the Domain model. In the EDM, it indicates whether the values of the attribute can be shared to external clients (value *Visibility::Public,* depicted as a green square) or not (value *Visibility::Private,* depicted as a red circle).

Moreover, in the concepts *mo::Record* and *mo::Track*, the designer defined three operations in order to retrieve instances of these concepts from a certain source and filter them according to a predefined condition. As an example, the *getAllRecordsByName* operation aims at searching the records whose name is similar to the name provided by the user. The type of this operation is *OperationType::ReadFilter* and the search condition is introduced by the designer. The name of the record to be found is provided as a parameter of the operation. In the same way, the designer defined the *getAllRecordsByDate* and *getAllTracksByName* operations, which aim at searching those records and tracks whose initial date and name are given by the users, respectively.

5.2.3 TWO MODELS, ONE DOMAIN

Given the characteristics of the Domain model and the EDM and the actions performed in each of the two initial tasks, the necessity of two models for the representation of the domain of the application might not seem appropriate. The Domain ontology and the EDM represent the domain knowledge using two domain specific languages whose abstract and concrete syntaxes might be similar. During the design of the S^{m}4RIA methodology, it could have been possible to address the design of these aspects of the application from other perspectives, e.g., creating only a model with the domain ontology (combining both models into one single model), creating the EDM first, etc.

It is possible to create the database structures from the domain ontology in the same manner as other methodologies such as SHDM propose (thus, transforming the model-driven methodology into an ontology-driven methodology, at least partially. Ontologies contain the knowledge of the domain needed to create a database schema from which the actual relational database, which is the most common type of database for Web applications, could be created. Taking into consideration this possibility, the ontology, or the EDM (and Extended Domain metamodel) in this case, would need to include some mechanisms in order to specify some features of the database, e.g., primary keys or type conversion mechanisms, among others. This could be achieved by extending the EDM or by modifying the S^{m}4RIA process in such a way that the Domain model could be created from the EDM (manually or automatically). With the modification of the process, the designer could reuse the efforts spent in the design of the Domain model and the transformation rules. Nevertheless, by applying this last solution the application would be bound to the ontology designed while the complexity of the resulting design process would not be affected drastically, since the actions performed would remain the same independently from the number of tasks or models.

The solution adopted separates the representation of the data objects in the database from the representation of the ontology, thus increasing the flexibility of the process. The current S^{m}4RIA process facilitates the definition of different domain ontologies (EDM) and the establishment of different mapping rules between the Domain model and them, which would lead to different views of the same data repository (different sets of ontology instances created from the same dataset). Moreover, it is totally compatible with the OOH4RIA design process, which also facilitates the adaptation of existing RIA applications to the SRIA approach or the transformation of legacy RIA applications into Linked Data repositories.

This solution is especially useful and recommendable in the development processes in which the stakeholders need to create a SRIA from their own legacy Web application (or from a legacy database) or need to develop a certain database structure (because it will be used by other applications). In other cases, model-to-model transformations can be applied to speed up this process.

5.3 Specification of the Navigational Concerns

After the specification of the EDM, in the following task of the S^{m}4RIA process (see Section 4.2.3, page 69), the server designer carries out the design of the Extended Navigation model (ENM). In this task, they establish the navigational paths through the application data, thus filtering the domain elements managed by the SRIA client components or external clients.

This model is conformant to the S^{m}4RIA Extended Navigation metamodel, which extends the OOH4RIA Navigation metamodel (see Section 4.2.4, page 71). The ENM, in the same manner as the OOH4RIA Navigational model, defines the navigation of the users from the designer's perspective in terms of two basic elements:

- *Navigational classes*, which are views of the data instances and operations associated to a class of the Domain model or a concept of the EDM. Navigational classes (depicted by three-compartment boxes in the OOH4RIA Navigational metamodel) can offer certain navigational attributes, which are views of attributes of the domain class or concept; and navigational operations, corresponding to operations of the domain class or concept. The initial navigation class is denoted by a small arrow on the top left corner of the box.
- *Navigational links*, which define the manner in which users can explore the data (moving from one navigational class to another) and invoke the operations offered. Navigational links, depicted as arrows in the OOH4RIA Navigational metamodel, can be classified into traversal links, which represent the transition between navigational classes, or service links, which represent the invocation of a navigational operation and whose origin is drawn as a square.

  The OOH4RIA Navigational metamodel organises the navigational classes into contexts, which are clusters of classes whose data can be visualised in the same client screenshot. A transition does not imply a change of context. Navigational links that do change the context are coloured in black. Otherwise, the arrows are depicted in white. Finally, transitions can be also activated automatically, in which case the metamodel illustrates them as dashed lines.

S^{m}4RIA includes new types of mechanisms in order to represent the retrieval of ontology instances from external sources and the navigation of the user through these data. Both aspects are closely related to the elements defined in the domain ontology and the ontologies imported in the EDM.

The ENM can be organised into different views, each for a different client role (e.g., *SRIA client*, *Semantic Web agent*, *B2B agent*, etc.) In this way, the application will offer service interfaces specialised for each type of client. For the development of SRIAs the designer must include two diagrams: one for the SRIA client, which will be used by the human users, and another for Semantic Web agents. The first one defines the manner in which users will interact with the application data and operations, while the second one specifies which ontology instances will be available for the Semantic Web agents through the SRIA Linked Data interface.

5.3.1 The Extended Navigational Meta-Model

The S^{m}4RIA Extended Navigation metamodel, which extends the OOH4RIA Navigation metamodel, described by Cachero et al. (Cachero et al., 2007), describes the abstract and concrete syntaxes of the ENM. It defines a collection of elements that facilitate the access to the content of external repositories of ontology instances and the process of sharing the data of the application as ontology instances.

In order to contextualise the contribution of the Extended Navigation metamodel, Figure 5.6 illustrates a diagram with the main components of the abstract syntax of the OOH4RIA EMOF Navigational metamodel as a class diagram. The metaclasses of this metamodel are coloured in dark grey, while the metaclasses of the OOH4RIA Domain model are coloured in red and their name include the package name (*conceptualView*).

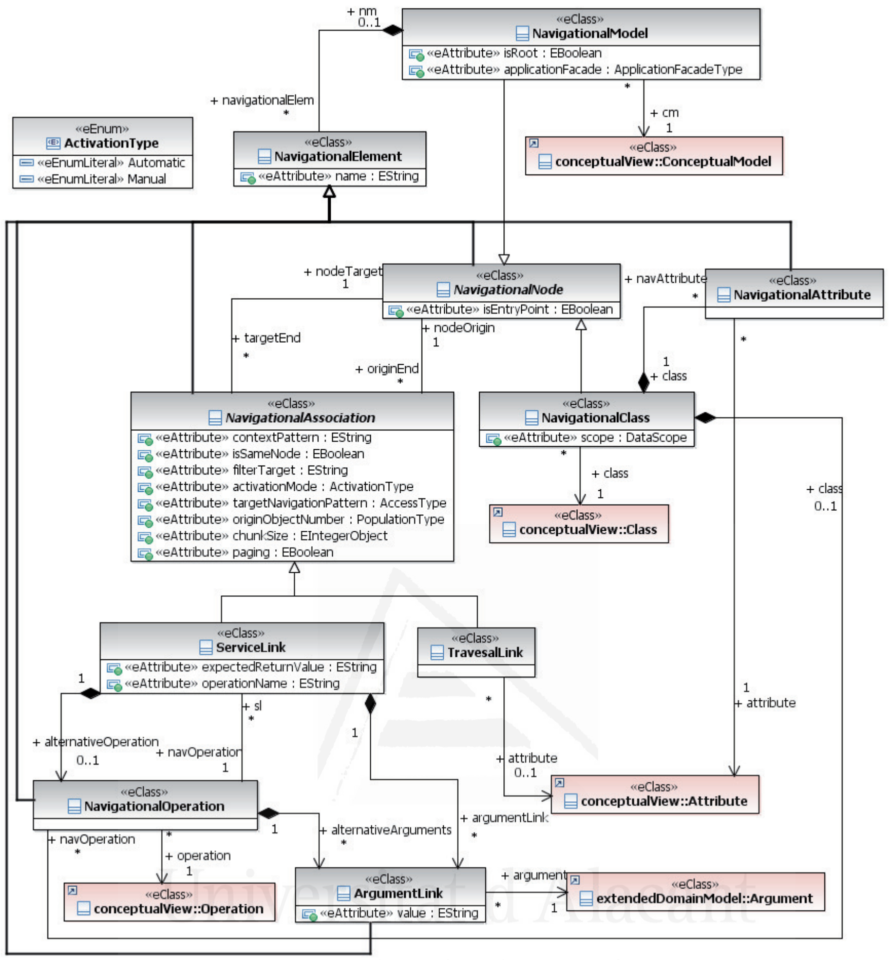

«eClass» NavigationalModel
«eAttribute» isRoot : EBoolean
«eAttribute» applicationFacade : ApplicationFacadeType
+ nm 0..1
+ navigationalElem *
+ cm 1
«eEnum» ActivationType
«eEnumLiteral» Automatic
«eEnumLiteral» Manual
«eClass» NavigationalElement
«eAttribute» name : EString
«eClass» conceptualView::ConceptualModel
«eClass» NavigationalNode
«eAttribute» isEntryPoint : EBoolean
+ nodeTarget 1
+ navAttribute *
«eClass» NavigationalAttribute
+ nodeOrigin 1
+ targetEnd *
+ originEnd *
1 + class
«eClass» NavigationalAssociation
«eAttribute» contextPattern : EString
«eAttribute» isSameNode : EBoolean
«eAttribute» filterTarget : EString
«eAttribute» activationMode : ActivationType
«eAttribute» targetNavigationPattern : AccessType
«eAttribute» originObjectNumber : PopulationType
«eAttribute» chunkSize : EIntegerObject
«eAttribute» paging : EBoolean
«eClass» NavigationalClass
«eAttribute» scope : DataScope
+ class 1
«eClass» conceptualView::Class
+ class 0..1
«eClass» ServiceLink
«eAttribute» expectedReturnValue : EString
«eAttribute» operationName : EString
«eClass» TravesalLink
1 + attribute
+ sl *
+ alternativeOperation 0..1
+ navOperation 1
+ attribute 0..1
«eClass» conceptualView::Attribute
«eClass» NavigationalOperation
+ argumentLink *
+ alternativeArguments *
+ navOperation *
+ operation 1
«eClass» conceptualView::Operation
«eClass» ArgumentLink
«eAttribute» value : EString
+ argument 1
«eClass» extendedDomainModel::Argument

Figure 5.6. Diagram of the OOH4RIA Navigational metamodel.

The S^{m}4RIA metamodel includes new types of external elements such as the External Navigational Class, which represents a view of data obtained from concepts and properties of the EDM. Moreover, it extends the OOH4RIA Navigational model with two new types of external links, which facilitate the combination of local and external knowledge. In the same way that Figure 5.6, Figure 5.7 depicts a class diagram with the abstract syntax of the S^{m}4RIA EMOF Extended Navigational Model. The figure shows the main elements of the metamodel and the connections to the Extended Domain metamodel (package *ExtendedDomainModel*) and OOH4RIA Navigational metamodel (package *NavigationalView*).

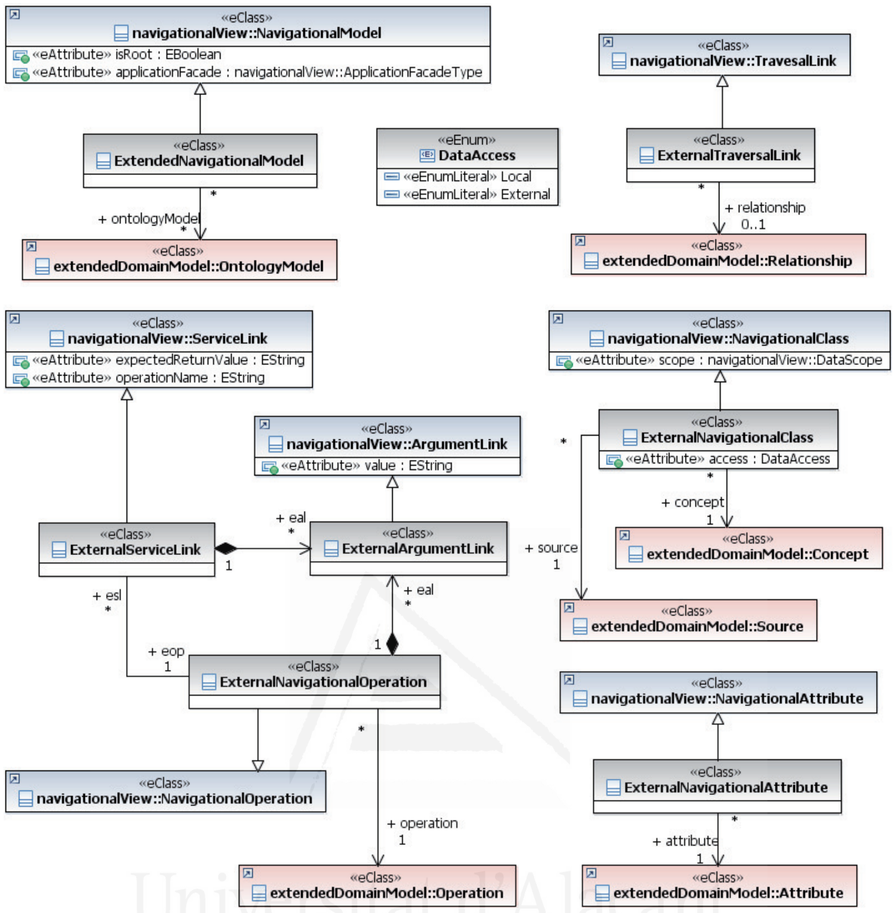
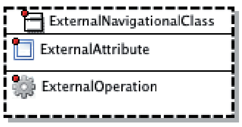

«eClass» navigationalView::NavigationalModel
«eAttribute» isRoot : EBoolean
«eAttribute» applicationFacade : navigationalView::ApplicationFacadeType
«eClass» ExtendedNavigationalModel
+ ontologyModel
«eClass» extendedDomainModel::OntologyModel
«eEnum» DataAccess
«eEnumLiteral» Local
«eEnumLiteral» External
«eClass» navigationalView::TravesalLink
«eClass» ExternalTraversalLink
+ relationship 0..1
«eClass» extendedDomainModel::Relationship
«eClass» navigationalView::ServiceLink
«eAttribute» expectedReturnValue : EString
«eAttribute» operationName : EString
«eClass» navigationalView::ArgumentLink
«eAttribute» value : EString
«eClass» ExternalServiceLink
«eClass» ExternalArgumentLink
+ eal
+ esl
+ eop
«eClass» ExternalNavigationalOperation
«eClass» navigationalView::NavigationalOperation
+ operation
«eClass» extendedDomainModel::Operation
«eClass» navigationalView::NavigationalClass
«eAttribute» scope : navigationalView::DataScope
«eClass» ExternalNavigationalClass
«eAttribute» access : DataAccess
+ concept
«eClass» extendedDomainModel::Concept
+ source
«eClass» extendedDomainModel::Source
«eClass» navigationalView::NavigationalAttribute
«eClass» ExternalNavigationalAttribute
+ attribute
«eClass» extendedDomainModel::Attribute

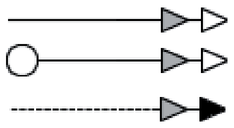
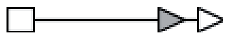
Figure 5.7. Diagram of the Extended Navigational Metamodel.

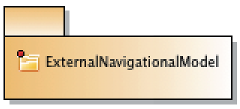
In order to facilitate the understanding of the diagram content, the metaclasses of the Extended Domain metamodel have been coloured in red while the ones of the Navigational metamodel in blue. Each of the elements defined in the metamodel (coloured in dark grey) is described in Table 5.3 with its graphical representation.

Table 5.3. Description of the new elements of the Extended Navigation Metamodel.

Metaelement	Description	Graphical Notation
ExternalNavigational-Class	This metaclass extends the *NavigationalClass* concept representing those classes whose data is obtained from ontology instances. The word "external" is introduced in order to highlight that the data of these classes will be	 ExternalNavigationalClass ExternalAttribute ExternalOperation

Metaelement	Description	Graphical Notation
	normally obtained from an external source. If the design needs to use local data, they can employ OOH4RIA navigational classes and retrieve these data from the database. Each *ExternalNavigationalClass* element must be associated to a concept and a source of the EDM. The *Source* element indicates the repository which the instances will be retrieved from and must be related to the *OntologyModel* element that contained the *Concept* element. The metaclass is depicted as a dashed box with three compartments: *a)* class name; *b)* navigational attributes; and *c)* navigational operations.	
ExternalNavigational-Attribute	This metaclass extends the concept of *NavigationalAttribute* representing those features of an *ExternalNavigationalClass* element that can be obtained. Each *ExternalNavigationalAttribute* element must be related to an Attribute element from the EDM, which, at the same time, must be contained in the concept associated to the container (i.e., the navigational class).	*See the ExternalNavigationalClass metaelement*
ExternalNavigational-Operation	This metaclass extends the concept of *NavigationalOperation* representing those operations of an *ExternalNavigationalClass* element that can be invoked as a service. Each *ExternalNavigationalAttribute* element must be related to an Attribute element from the EDM, which, at the same time, must be contained in the concept associated to the container (i.e., an external	*See the ExternalNavigationalClass metaelement*

Metaelement	Description	Graphical Notation
	navigational class).	
ExternalTraversalLink	This metaclass is an extension of the *TraversalLink* metaclass and represents a link between a *NavigationalClass* element and an *ExtendedNavigationalClass* element, or between two *ExtendedNavigationalClass* elements. In those cases where the two *ExtendedNavigationalClass* elements connected by a link are associated to *Source* elements of different *OntologyModel* elements, it is needed to manually define a rule that transform the origin object into an equivalent object using the second *OntologyModel* element. In this way, it will be possible to navigate through the instances of the second Source element or to extend the information of the object. The concrete syntax of these elements is similar to the one of the TraversalLink elements. In this case, the traversal link is depicted as a double arrow in which the internal arrow is always grey and the external one depends on whether the user navigates between contexts (black arrow) or stays in the same context (white arrow). Automatic links, i.e., those whose invocation does not require the interaction of the users, are represented using dashed lines. If the traversal link is associated to a relationship of the EDM the origin of the link is depicted as a white circle.	
ExternalServiceLink	This metaclass extends the *ServiceLink* metaclass and represents the invocation of an *ExternalNavigationalOperation* element and the transition between	

Metaelement	Description	Graphical Notation
	one class and another. *ExternalServiceLink* elements can only be established between *ExternalNavigationalClass* elements. The concrete syntax of the ExternalServiceLink elements is similar to the ExternalTraversalLink ones. The main difference is the origin of the arrow, which, in this case, is depicted as a white square.	
ExtendedNavigational-Model	This metaclass extends the *NavigationalModel* metaclass including new S^{m}4RIA metaclasses already described. An *ExternalNavigationalModel* element can contain other submodels of the same type. The graphical notation of this element is similar to the UML Package.	

The constraints required to maintain the consistency of the model and the relationship between models (e.g., avoid that navigational classes include navigational attributes from different concepts) are defined in a collection of OCL constrains. Table 5.4 introduces a summary of the OCL constrains defined over the metamodel.

Table 5.4. Summary of the OCL constraints of the Extended Navigational metamodel.

Name	Description / OCL Contraint
Source&Model	The *Source* and the *OntologyModel* elements of an *ExternalNavigationalClass* element must be also associated.
	`context ExternalNavigationalClass inv: self.concept.model.sources->target->contains( self.source )`
OnlyValidOpAttr	All the navigational attributes and operations of an *ExternalNavigationalClass* element must be of types *Extended-NavigationalAttribute* and *ExtendedNavigationalOperation*, respectively.

Name	Description / OCL Contraint

	`context ExternalNavigationalClass`
	`inv:`
	`        self.navAttribute->forAll(a | a.OclIsTypeOf(`
	`        ExtendedNavigationalAttribute ))`
	`inv:`
	`        self.navOperation->forAll(o | o.OclIsTypeOf(`
	`        ExtendedNavigationalOperation ))`

AttrConsistency	The attribute of the Domain model and the attribute of the Extended Domain Model to which is related an *ExternalNavigationalAttribute* element must be related each other.

	`context ExternalNavigationalAttribute`
	`inv:`
	`        self.domAttribute = null ||`
	`        self.domAttribute =`
	`        self.attribute.domAttribute`

OnlyRelatedAttr	An *ExternalNavigationalClass* element can only contain *ExternalNavigationalAttribute* elements created from the attributes of the concept (from the Extended Domain Model) to which is related.

	`context ExternalNavigationalAttribute`
	`inv:`
	`        self.attribute.concept =`
	`        self.navClass.OclAsType(ExtendedNavigational`
	`        Class).concept`

ValidExternalTraversalL ink	An *ExternalTraversalLink* element can only be created from a relationship existing between the concepts associated to two *ExternalNavigationalClass* elements.

	`context ExternalTraversalLink`
	`inv:`
	`        self.relationship.target =`
	`        self.nodeTarget.OclAsType(ExtendedNavigation`
	`        alClass).concept`
	`        &&`
	`        self.relationship.concept =`
	`        self.nodeOrigin.OclAsType(ExtendedNavigation`
	`        alClass).concept`

5.3.2 The Extended Navigational Model of the SNS Case Study

Following the development of the case study, in this task, the server designer creates the Extended Navigational Model. As mentioned in the beginning of this section, the designer needs to define one navigational diagram per type of client. In this case, they need two diagrams: one for the users that access through the SRIA client and another for the software agents that employ the Linked Data interface and the HTML interface.

The diagram for the SRIA client of the Extended Navigational Model is depicted in Figure 5.8.

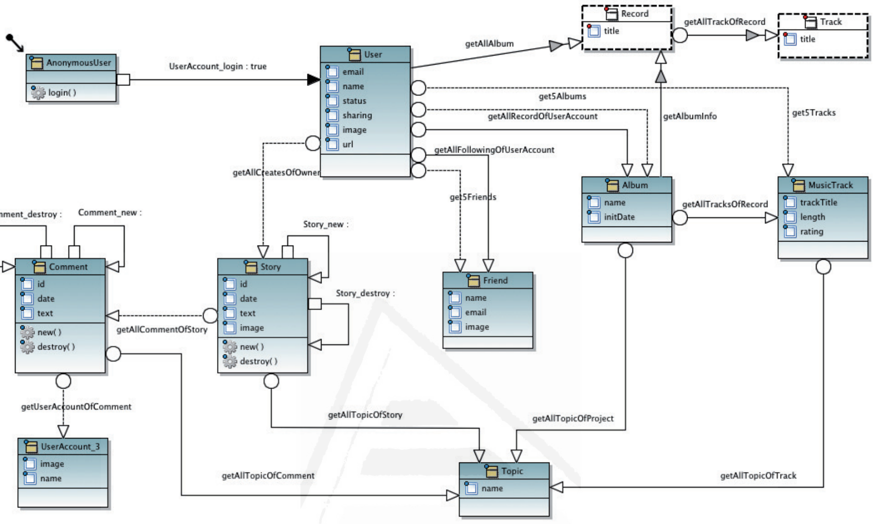


Figure 5.8. Diagram of the Extended Navigational Model for the SNS case study.

In this model, the initial class is the *AnonymousUser* navigational class, which offers the *login* navigational operation that can be used in order to authenticate users using the name of their user account and a password. The *UserAccount_login* service link represents the manual invocation of the login operation (e.g., using a form) and the transition to the user wall (class *User*, which also implies a change of context) if the result of the operation is set to *true*. During the process of authentication, the application also needs to retrieve the data of the user, which is the target of the service link, since the *login* operation only returns a boolean value indicating whether the password corresponds to the user account or not. In order to solve this issue, the designer defines an Alternative Target Getter (ATG) operation in the *UserAccount_login* service link. This operation can be defined when the main operation invoked does not return an object (instantiation of a Domain Model class) and it is not represented graphically. In this case, the designer specifies the invocation of the *ReadOID* operation from the User domain class, which, given a valid Object Identification (OID) value, returns all the data of the object of the *User* class identified by that value.

After the process of authentication, the user can visualise their personal information in the wall. Moreover, with the automatic invocation of the *get5Friends*, *get5Albums* and *get5Tracks* traversal links, the user can explore three five-element random lists with their friends, albums and tracks of the application, respectively. The number of elements returned by a traversal link can be set with the *Chunk* metaproperty. With the *getAllCreatesOfOwner* automatic link, users can obtain with no change of context their corresponding walls, with a list of stories and comments (class *Comment*), as well as the information of the user who created the comment (class *UserAccount_1*) which are retrieved with the *getAllCommentOfStory* and *getAllUserAccountOfComment* automatic traversal links. The designer also offered the possibility to create and delete stories and comments using the *New* and *Destroy* operations of the *Story* and *Comment* classes. The *New* methods create new objects of the same type that the navigational class using the navigational attributes as parameters. The *Destroy* methods delete an object from the system based on its OID value.

Furthermore, users can visualise a complete list of their friends and albums using the *getAllFriends* and *getAllRecordOfUserAccount* traversal links. These links are associated to two association roles and represent the retrieval of all the users and albums associated to the first user. Retrieving all the possible objects of a class might result in a loss of performance in the application. Therefore, the OOH4RIA Navigational metamodel includes a property in order to indicate that the objects will be retrieved following a process of pagination (adding two extra parameters to the operation: the offset and the number of objects).

All the local elements of the SNS are related to a tag (*Tag* navigational class), which is a word or collection of words that characterise the content of an object and can be used to search similar elements. The tags of each element are automatically retrieved when the user obtains the information about each object.

Moreover, users will be able to search information about artists, albums and tracks stored on MusicBrainz using the *getAllAlbum* and *getAlbumInfo* external traversal links. The *Record* external navigational class is a view of the Record concept from the *MusicOntology* ontology model and it specifies which information from the record (or album) must be retrieved from the MusicBrainz source (see the EDM). The relation between the navigational class and the source is not depicted in the diagram.

Using the *getAllAlbum* external link, the designer specifies the retrieval of all the instances of the *Record* concept and their title (associated to the *title* property from Dublin Core) using the strategy of pagination. The designer also defined the *getAllTrackOfRecord* external link that facilitates the navigation from the data of the Record to the data of the Track (*Track* external navigational class, defined from the *Track* concept of the *MusicOntology* ontology model and the *MusicBrainz* source). This link is associated to the *tracks* property of the *MusicOntology* ontology model.

With the *getAlbumInfo* external link, the designer specified a link that shows the information of an album stored in MusicBrainz. In order to do so, the designer defined a rule that map the name of the album into the name of the record using OCL. Although the *Album* and *Record* navigational classes could be associated (directly or indirectly) to the *Record* concept from the *MusicOntology* model, the objects from two different repositories do not usually share the same method for the creation of identifiers (e.g., URIs). As a consequence, it is not possible to know beforehand whether two objects are equivalent only by checking their URIs (they might not contain any axiom of equivalency as well). Designers need to specify the manner in which the application should seek equivalent objects. In this link, the designer included a condition in order to ensure that the titles of the origin and target objects are equal. This condition varies depending on the *Concept* and *Source* elements.

The navigational elements used for the management and administration of the application are modelled in different packages (*ExtendedNavigationalModel* elements). For instance, the classes for the management of the artists' profiles are included in the *ArtistManagement* package.

Once the first diagram is completed, the server designer specifies a new diagram in the Extended Navigational Model for those software agents that will access the data through the Linked Data service or the HTML view of the SRIA (see Figure 3.2, page 50). This second diagram employs the elements of the Extended Navigational metamodel in order to constrain the domain ontology instances that will be generated for each navigational class by the final software modules. Figure 5.9 illustrates a fragment of the navigational diagram for software agents.

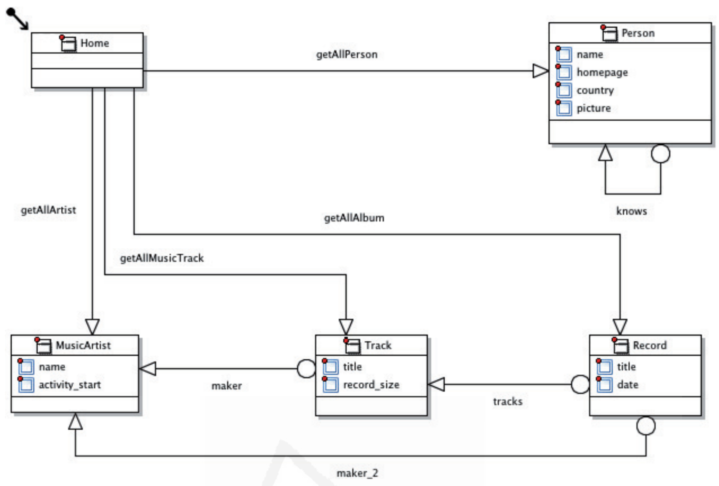


Figure 5.9. Diagram (partial) of the Extended Navigational Model for Semantic Web agents.

In this model, the designer employs a collection of External Navigational classes related to the *LocalSource Source* element of the EDM. According to this model, the system will grant access to part of the information stored in the user profiles as FOAF instances by means of the Person navigational class, associated to the FOAF Person concept, and the *getAllPerson* traversal link. However, in order to protect the privacy of the users, the access to the data is limited by an OCL constraint in the link:

```
Context Home::get_users
pre:
   self.target.sharing = "open";
```

With this constraint, the generation of ontology instances will be performed only from those user accounts whose *sharing* property was set to "open". The *sharing* property was included as a private property in the EDM, indicating that the content of this property will not be shared but the property can be used for other purposes during the design process. The values of this property can be managed by human users in the other view of the model depending on their preferences using custom operations.

Furthermore, the application will share information about the artists, albums and tracks it contains as MusicOntology instances using the Artist, Record and Track external navigational classes respectively, which are associated to the MusicOntology's Artist, Record and Track concepts and related to the LocalSource element.

Using concepts from external ontologies, instead of local ones, for the generation of ontology instances, the opportunities of reutilisation of these instances notably increase. They could be assimilated by MusicBrainz or by other social network. Although there is no standard in this field, the use of an ontology widely instantiated in the Linked Data cloud can obviously increase the adoption of the instances produced.

5.4 Conclusions

This chapter described in detail the first activity of the S^{m}4RIA process "*Design the SRIA server*", in which the server and the ontology designer capture (from the stakeholders' requirements) and represent the main information needed for the development of the SRIA server in three models.

The activity groups all actions into three tasks in which, as a result, a new model is obtained: the Domain model, the Extended Domain Model and the Extended Navigation Model, respectively. Each section of the chapter explained the actions performed, the information contained in each of the models and the abstract and concrete syntaxes of each metamodel. In order to facilitate the understanding of the process and the models, the chapter explained the development of the SNS case study using the S^{m}4RIA models.

The obtained models are the input resources for the next activity, explained in detail in the following chapter. The next chapter addresses all the issues concerning the design of the SRIA client.

Chapter 6. DESIGNING THE CLIENT COMPONENTS OF A SEMANTIC RICH INTERNET APPLICATION

SRIA clients visualise the data (local data structures and external ontology instances) and invoke the operations offered by the RIA server by means of asynchronous communication processes, i.e., clients do not get blocked while waiting for a response from their servers. Unlike traditional Web applications, RIAs follow a "simple page application" strategy (Mesbah and Van Deursen, 2007), in which their user interface (UI) consists of a single page with a set of stateful widgets, i.e. UI structural components for the representation of data and the interaction with the users. Moreover, (S)RIA UIs are driven by events. Users can trigger a list of events specific for each type of widget and, depending on the triggered events, the SRIA client performs different actions which might involve the invocation of a server service or just a local modification of the interface. As can be appreciated, there are complex interaction dependencies between widgets and users, as well as between the (S)RIA server and client widgets, which should be considered during the design and development of the application.

The S^{m}4RIA methodology addresses these issues in the second activity of the process (please, check the SPEM2 diagram in Figure 4.2, page 70) reusing and extending the two OOH4RIA RIA-specific models (PSM) for the representation of UIs:

(i) the *Extended Presentation Model*, created from the OOH4RIA Presentation model, which represents the structure of the user interface as a collection of different panels and widgets based on a WYSWYG visualisation (*What-You-See-is-What-You-Get*), thus allowing designers with no expertise in programming to define a RIA UIs; and

(ii) the *Extended Orchestration Model*, based on the OOH4RIA Orchestration model, which captures the interactions between the UI widgets and the rest of the system by means of a collection of *Event-Condition-Action* (ECA) rules.

Specifically, this activity is composed of six tasks performed by three actors, i.e., the UI designer, the ontology designer and a M2M transformation engine. The output artefacts of the second activity are the two aforementioned models. The main aims of this activity are the following:

a) To design the structure of the user interface in terms of screenshots, panels and widgets and their properties (e.g., size and position).
b) To specify the visual appearance of each widget (background colour, foreground colour, font options, etc.)
c) To specify the behaviour of the user interface: event management, widget modifications, panel modifications, etc.
d) To design the access to the local data of the application by means of the invocation of the SRIA server services from the elements of the user interface.
e) To design the access to external ontology instances from the repositories of the Linked Data cloud by means of the invocation of the SRIA server services.
f) To specify ontology-based annotations over static widgets or panels.

The S^{m}4RIA methodology reuses the main tasks of this process from the OOH4RIA process and includes two new tasks that associate the SRIA UI models to the external knowledge sources defined in the EDM and ENM: the "*Enrich Presentation Model*" and "*Enrich Orchestration Model*" tasks. These tasks could be integrated in the "*Design Presentation Model*" and "*Design Orchestration model*" tasks, respectively. However, in this case, in order to distinguish between the actions already performed in the OOH4RIA process and the new actions included in S^{m}4RIA, the adopted solution was to create two new tasks. In these tasks, the ontology designer can specify ontology-based annotations to the UI widgets by means of three patterns that were not considered in OOH4RIA (see Figure 6.1):

1) To establish a relationship between the SRIA UI components and the external navigational classes in order to gather information from external ontology instances using the services provided by the SRIA server;
2) To connect UI actions from the Extended Orchestration model to external navigational links, which define the retrieval of ontology instances (from external or internal sources) on demand; and
3) To define direct annotations from UI widgets to ontology elements, thus allowing the retrieval of information about these UI elements.

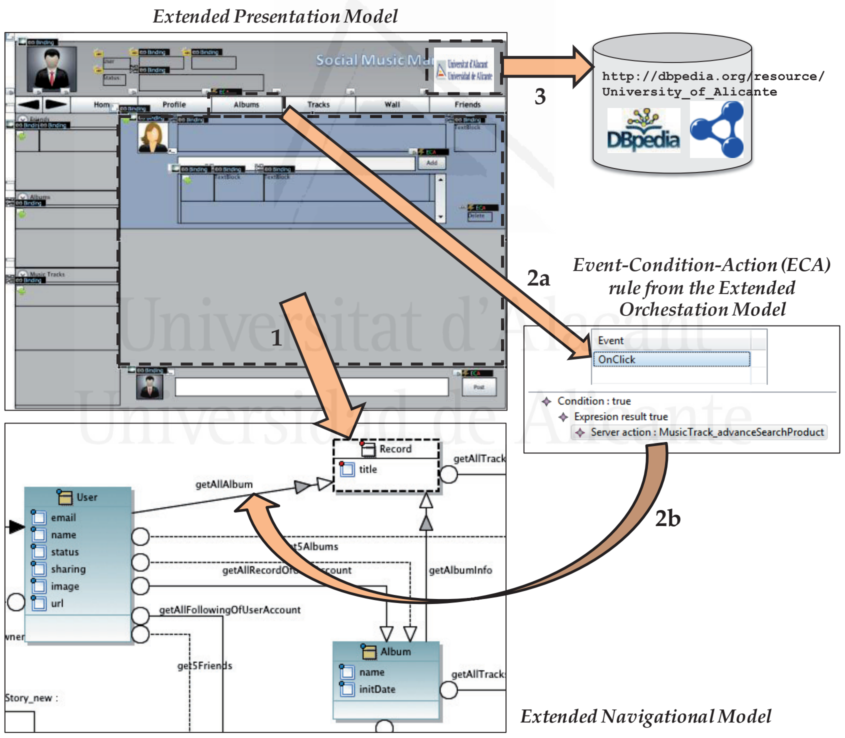


Figure 6.1. Patterns of extension of the Presentation and Orchestration models.

This chapter describes in detail the tasks of the second activity and the models designed. The subsequent sections will explain the process followed so as to create each of the resulting models, grouping the actions of different tasks and including a detailed description of the metamodels employed. In order to facilitate the understanding and analyse the performance of the process, the actions explained will be applied to the case studies explained in Sections 3.4.1 and 3.5.2 (pages 54 and 60, respectively): the social network site and the RI@BI application.

6.1 DEFINING THE STRUCTURE OF THE USER INTERFACE

In the beginning of the second activity of the S^{m}4RIA process, the *ExtNav2Press* M2M transformation can create a mock-up of the Extended Presentation model from the Extended Navigational Model, which should be subsequently completed by the UI designer. This is an optional task since the Extended Presentation model can be also defined from scratch. The transformation can create a presentation model with a basic UI, which would contain the elements needed to manage the information and the operations represented in the ENM (e.g., for a navigational operation, it would create a new form). However, there might be several valid presentation models (considering structure and appearance) from a single navigational model depending on the needs of the stakeholders, which might decelerate the process of design. Moreover, there are some aspects such as the appearance that cannot be inferred from the navigational models. The use of this transformation can accelerate the design process of those clients whose requirements do not change between applications considerably, such as the interfaces for the administration of the application.

Another aspect that should be highlighted is the fact that the structure and appearance of the user interface could be defined directly from the stakeholders' requirements before the ENM is completed, thus partially removing the dependency between the ENM and the UI models. However, this dependency cannot be totally removed since the data contexts defined in the ENM model constrain the information shown in a screenshot and the server operations invoked. Using this strategy, both models could be specified at the same time, thus increasing the flexibility of the process, but there is a risk of model inconsistency that should be resolved by the server and UI designers. This chapter explains the original activity considering that the design of the Extended Presentation Model is performed from scratch (based on the stakeholders' needs) once the ENM model is completed.

As mentioned before, the UI designer specifies the EPM from the stakeholders' requirements and the ENM. This process can be divided in two stages: the creation of the interface and the establishment of connections between the EPM and the ENM. In the first stage, the UI designer actually builds the UI of the SRIA client, defining the structure and the visual representation of the UI. The basic components of the EPM are the following:

a) *Widgets*, which are the basic components in the user interface. They can be classified into two groups: data representation widgets, used to visualise data (e.g., a label), and interaction widgets, employed for the interaction with the users (e.g., a button).
b) *Panels*, which are groups of widgets arranged according to their own rules (e.g., a stack panel or a canvas). The relationship of containment in a panel is transitive.
c) *Screenshots*, which are representations of the complete view that is shown to the users. These elements contain panels and widgets arranged according to different criteria. RIAs usually contain one single screenshot. However, it is also possible to create multi-page RIA with several screenshots.

The design of the EPM commences with the definition of an initial screenshot, in which the UI designer includes a collection of widgets or panels, containing other widgets at the same time. The designer can arrange the widgets within the panels or the screenshot depending on their type. For instance, the stack panel constrains the position of the widgets creating a stack while the canvas widget allows the designer to include a widget where needed. The designer can create as many screenshots as they need.

According to the Model-Driven Architecture (Object Management Group, 2003), the Extended Presentation Model and the Extended Orchestration Model are platform-specific models, which means that they include elements specific for the design of a certain component of the application taking into consideration the final implementation technology. The EPM aims at representing a view of the final SRIA UI. However, the EPM components (panels and widgets), the actions that can be performed over them and the appearance options can change among RIA technologies. Therefore, in order to obtain a reliable representation of the UI, it is necessary to adapt the metamodel of the EPM to the particularities of each technology. The OOH4RIA Presentation and Orchestration models were adapted to the design of Silverlight user interfaces. As a consequence, the EPM and the EOM, as extensions of both models, are adapted to the same platform.

In the second stage, which might be performed in parallel to the first one, the UI designer connects the UI components to the ENM elements, thus establishing the areas for data visualisation and the type of objects shown in each one. There are three types of relationship between the EPM and the ENM models (see a schema of them in Figure 6.2):

1) *Context relationship*. The first relationship defines the type of objects (or ontology instances) visualised in a screenshot or a panel. While a panel can visualise a collection of objects, the screenshot must be associated to a single object. Moreover, a screenshot can be only related to an entry point or to the destination (a navigational class) of a link that implies a change of navigational context.
2) *Navigation relationship*. This relationship associates a panel to a navigational link in such a way that it indicates that the context of the panel is a subcontext of the container and the manner in which the objects can be retrieved and visualised in case that the link is traversal and automatic. For manual links and service links, the invocation will be specified in the Extended Orchestration Model.
3) *Binding relationship*. The Binding relationship establishes the data of an object shown in a widget. It relates a widget to an attribute of one of the contexts which the widget is contained in. Since the property of containment in a panel is transitive, a widget can be bound to any property of the contexts associated to the panels (or the initial screenshot) it is contained in.

Figure 6.2 introduces a schema with different examples of the three types of relationships between models. The schema illustrates a part of a hypothetic EPM (left-hand side) and the links to a part of the ENM, introduced in Section 5.3.2 (page 107). As can be appreciated, the EPM is composed by two screenshots.

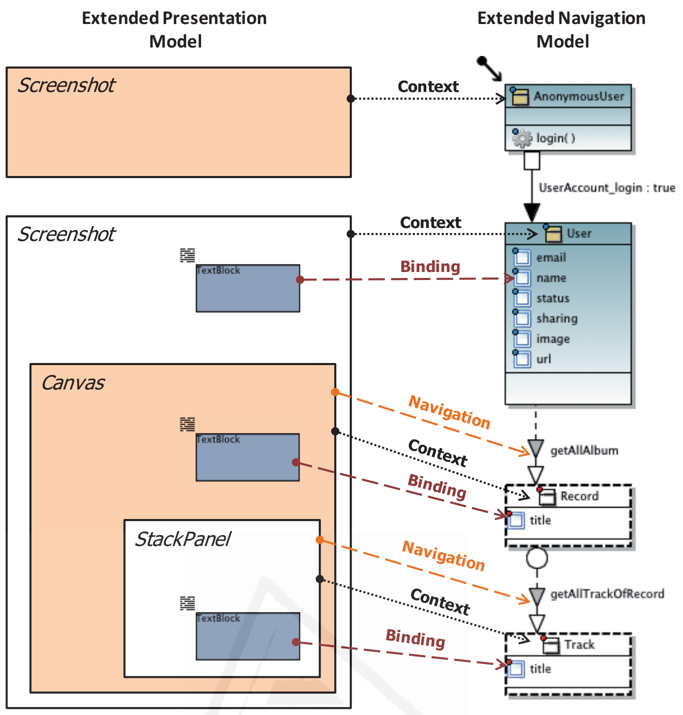


Figure 6.2. Different types of relationship between the Extended Presentation Model and the Extended Navigational Model.

The first screenshot is associated to the *AnonymousUser* navigational class, which indicates that the operations of the navigational class will be invoked from this screenshot. The UI designer should specify the widgets needed to create a form that could invoke the operation and the process of invocation will be defined in the EOM.

The second screenshot contains two panels, i.e., a canvas and a stack panel (from Silverlight) and three widgets (*TextBlock* widgets from Silverlight, similar to a label) included in the available containers. This screenshot is linked to the *UserAccount* navigational class, which facilitates that the TextBlock widget could be associated to any of the navigational attributes of the mentioned class. The *Canvas* panel is associated to the *Record* external navigational class and the *getAllAlbum* external traversal link, thus indicating that the context of this class is obtained by means of the link. The *TextBlock* widget included in the *Canvas* panel could be bound to any navigational attribute of the *Record* or *UserAccount* class.

The mentioned elements and the relationships are formalised in the Extended Presentation metamodel, whose abstract and concrete syntaxes are introduced in the next subsection.

6.1.1 THE EXTENDED PRESENTATION METAMODEL: CONCRETE AND ABSTRACT SYNTAXES

The Extended Presentation Model is conformant to the Extended Presentation Metamodel, which extends the OOH4RIA Presentation metamodel. This extension includes the following elements:

1. new links in order to associate the UI widgets with the external navigational classes and external traversal and service links;
2. new types of widgets for the representation of aggregation of data (e.g., maps, charts, etc.);
3. the concept of ontology-based annotation, which links a UI element to a URI pointing at an ontology element or an element of the EDM.

The Extended Presentation Metamodel contains the mechanisms for the representation of the elements of the user interface (widgets, containers and screenshots) in a platform-specific manner, i.e., including elements specific for a RIA technology, in this case, Silverlight. In order to facilitate the adaptation of the OOH4RIA Presentation model to different technologies and the connections between the Presentation and Orchestration models, this metamodel is divided in two parts:

1. *Abstract components*. These are the core elements of the metamodel, which include the generic (or abstract) definition of the *screenshot*, *widget* and *container* components and the relationships with elements of other models. These part of the metamodel also represent a collection of common widgets and containers, i.e., found in several technologies, and their properties.
2. *Platform-specific components* (in this case, Silverlight). These are the elements of the metamodel that depend on the technology chosen in the development process. This part of the metamodel is composed of the Silverlight-specific widgets and containers and the relationships between them.

The diagram illustrated in Figure 6.3 represents the main elements of the Extended Presentation metamodel. The platform-specific metaclasses (in this case, Silverlight-specific metaclasses) are named with the prefix “SL” (which stands for “Silverlight”). For a more detailed representation of the metamodel, which includes all the possible elements, please check the diagrams included in Annex E (page 273).

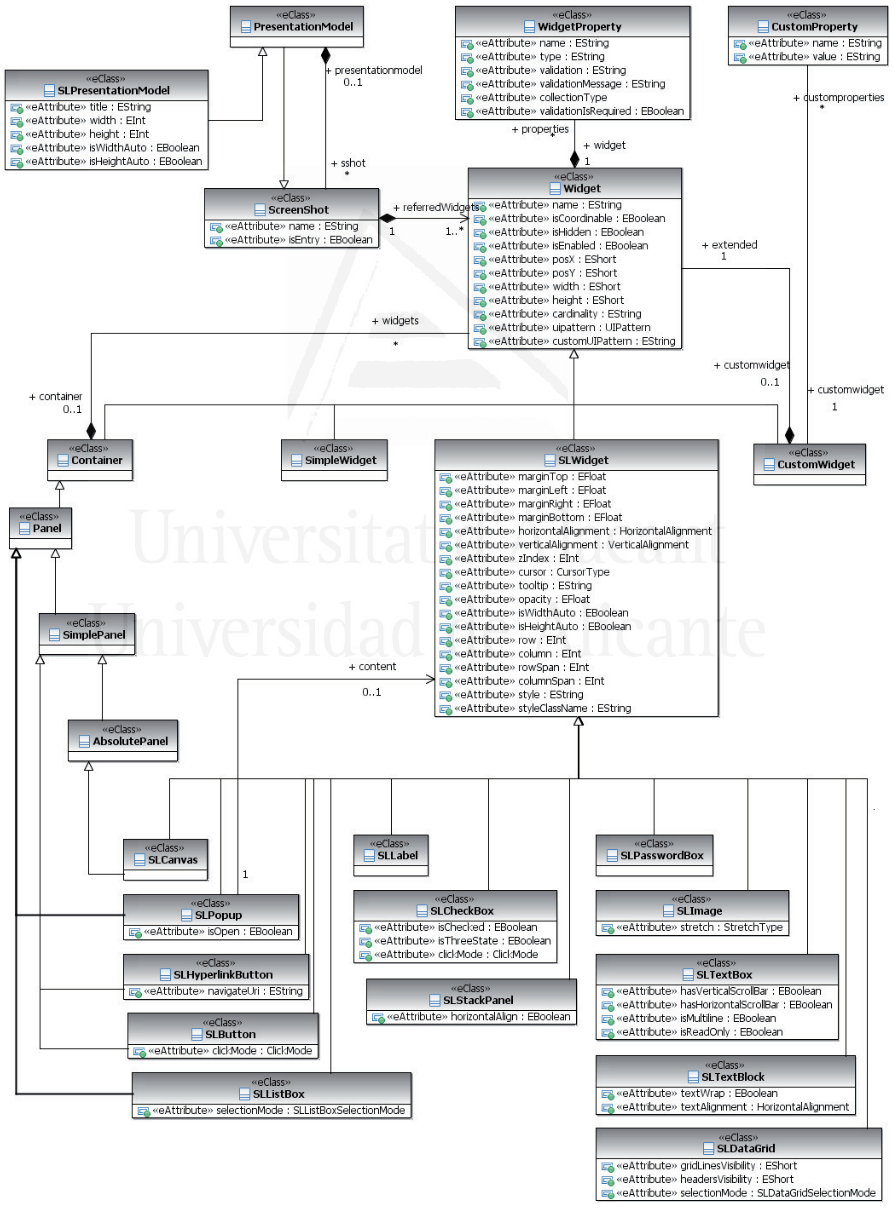


Figure 6.3. Main elements of the Extended Presentation Model.

Figure 6.4 depicts a schema with the connections between the elements of the Extended Presentation metamodel and the elements of the Extended Domain metamodel (*extendedDomainModel* namespace) and the Extended Navigational metamodel (*navigationalView* namespace). This diagram introduces the elements needed to represent the three relationships (*Context*, *Navigation* and *Binding*) that connect the UI elements to the elements of the Extended Navigational Model as the last section introduced. In the diagram, the *navContext* link corresponds to the *Context* relationship, the *navigation* link to the *Navigation* relationship and the *WidgetPropertyBinding* metaclass and the *navType* link to the *Binding* relationship. It is worth noticing that the *Container* metaclass is not considered in this diagram because it was created using the *Composite* pattern as a subtype of the *Widget* metaclass that contains other widgets.

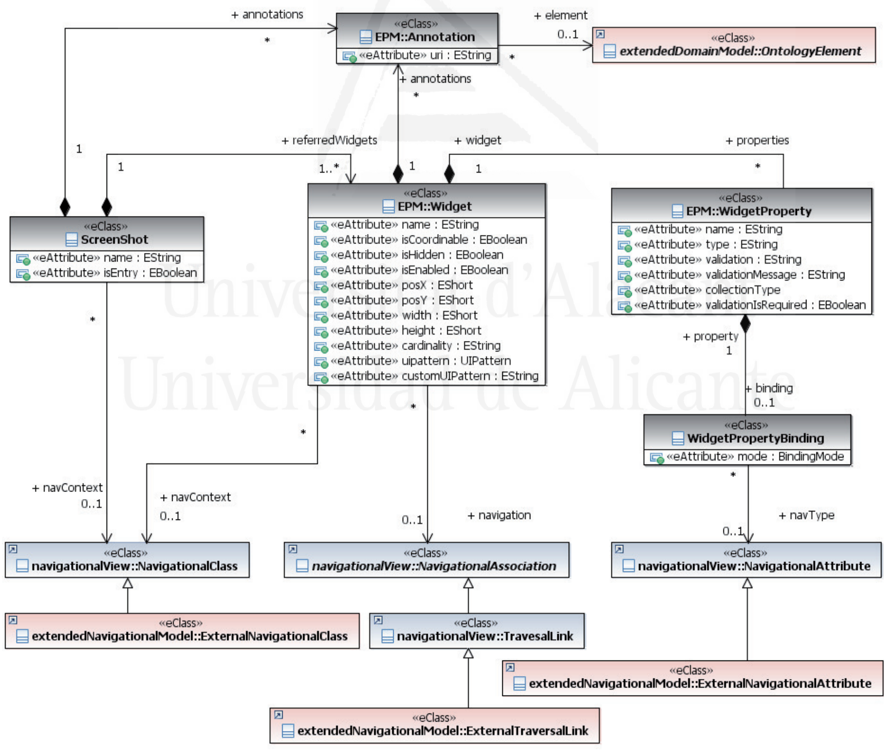
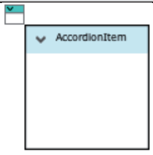
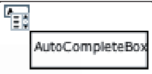
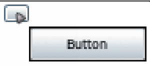
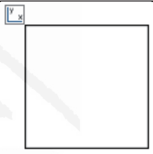
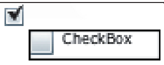
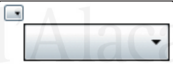
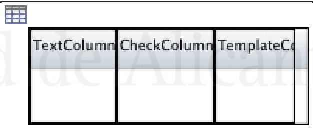
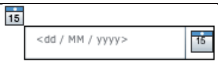
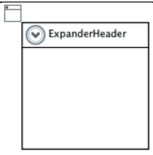
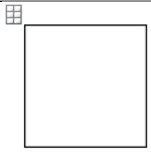
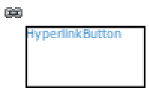

+ annotations
*
«eClass»
EPM::Annotation
«eAttribute» uri : EString
+ element
0..1
*
«eClass»
extendedDomainModel::OntologyElement
+ annotations
*
1
+ referredWidgets
1..*
1
+ widget
1
+ properties
*
«eClass»
ScreenShot
«eAttribute» name : EString
«eAttribute» isEntry : EBoolean
«eClass»
EPM::Widget
«eAttribute» name : EString
«eAttribute» isCoordinable : EBoolean
«eAttribute» isHidden : EBoolean
«eAttribute» isEnabled : EBoolean
«eAttribute» posX : EShort
«eAttribute» posY : EShort
«eAttribute» width : EShort
«eAttribute» height : EShort
«eAttribute» cardinality : EString
«eAttribute» uipattern : UIPattern
«eAttribute» customUIPattern : EString
«eClass»
EPM::WidgetProperty
«eAttribute» name : EString
«eAttribute» type : EString
«eAttribute» validation : EString
«eAttribute» validationMessage : EString
«eAttribute» collectionType
«eAttribute» validationIsRequired : EBoolean
+ property
1
+ binding
0..1
«eClass»
WidgetPropertyBinding
«eAttribute» mode : BindingMode
*
*
*
*
+ navContext
0..1
+ navContext
0..1
0..1
+ navigation
+ navType
0..1
«eClass»
navigationalView::NavigationalClass
«eClass»
navigationalView::NavigationalAssociation
«eClass»
navigationalView::NavigationalAttribute
«eClass»
extendedNavigationalModel::ExternalNavigationalClass
«eClass»
navigationalView::TravesalLink
«eClass»
extendedNavigationalModel::ExternalNavigationalAttribute
«eClass»
extendedNavigationalModel::ExternalTraversalLink

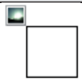
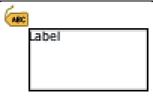
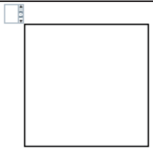
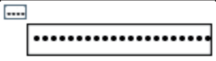
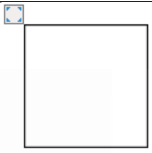
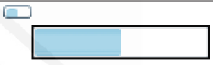
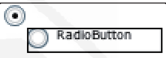
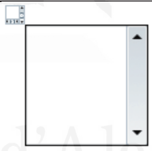
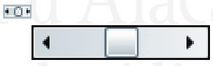
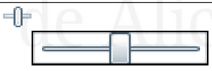
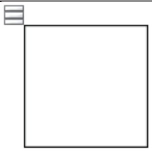
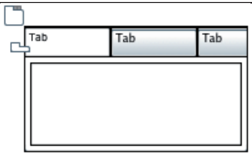
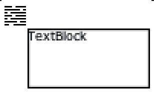
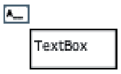
Figure 6.4. Connections between the EMOF Extended Presentation metamodel and other S^m4RIA metamodels.

It is also worth noticing that, since the EPM is a WYSIWYG model, only the platform-specific elements of the metamodel have a graphical notation, which will be dependent on the final technology as well. Table

6.1 contains the graphical notation of each of the platform-specific metaclasses represented in Figure 6.3 and Annex E .

Table 6.1. Graphical notation of the platform-specific metaclasses of the Extended Presentation Model.

Metaclass	Graphical Notation
SLAccordion	
SLAccodionItem	*See SLAccordion metaclass*
SLAutoCompleteBox	
SLButton	
SLCanvas	
SLCheckBox	
SLComboBox	
SLDataGrid	
SLDataGridColumn	*See SLDataGrid metaclass*
SLDateBox	
SLExpander	
SLGrid	
SLHyperlinkButton	

Metaclass	Graphical Notation
SLImage	
SLLabel	
SLListBox	
SLPasswordBox	
SLPopup	
SLProgressBar	
SLRadioButton	
SLScrollViewer	
SLScrollBar	
SLSlider	
SLStackPanel	
SLTabControl	
SLTabItem	*See SLTabControl metaclass*
SLTextBlock	
SLTextBox	

In order to complete the description of the metamodel, Table 6.2 introduces some of the main OCL constraints specified, which address issues related to the consistency of the information stored by the model.

Table 6.2. Summary of the OCL constraints of the Extended Presentation Model.

Name	Description / OCL Contraint

DifferentAnnotations	All the *Annotation* elements of a *Widget* element must be different.
	`context Widget`
	`inv:`
	`self.annotations->forAll(a | self.annotations->select(a2 | a.uri = a2.uri)->size() = 1)`

ConsistentAnnotation	When the annotation is linked to an element of the Extended Domain Model, the URI of the Annotation element and the element of the EDM must be consistent.
	`context Annotation`
	`inv:`
	`self.element = null || self.uri = self.element.uriBase + '/' + self.element.name`

NotBindingWithoutContext	The relationships of binding cannot be established unless the widget or one of its containers has a context.
	`context Widget`
	`inv:`
	`((self.navContext = null || self.ascendants()->navContext->isEmpty()) && self.properties-> binding->size() = 0)) || self.navContext <> null`

BindingAssociatedToContext	The relationships of binding between a navigational attribute and a *Widget* element must be only established with the navigational attributes of the context (navigational class) of the *Widget* element or any of the widget containers
	`context Widget`
	`inv:`
	`self.navContext.navAttributes->union( self.ascendants()->navContext->navAttributes )->includesAll( self.properties->binding-> navType )`

ExternalLinkTarget	The target of an external link (i.e., *ExternalTraversalLink* or *ExternalServiceLink* elements) must be an *ExtendedNavigationalClass* element.

Name	Description / OCL Contraint

	`context Widget`
	`inv:`
	`        self.navigation = null ||`
	`                (self.navigation.oclIsTypeOf(`
	`                ExternalLink)`
	`                && self.navContext.oclIsTypeOf(`
	`                ExtendedNavigationalClass))`

The next section will explain the design of the EPM using a real case study, i.e., the design of the UI of a social network site, following the example described in other sections.

6.1.2 An Extended Presentation Model for the Social Network Site

In the beginning of the second activity, the UI designer creates the EPM in order to represent the structure of the user interface of the SRIA (including the main features of the UI widgets) and links the UI components to the elements of the ENM, defined in the last activity. As a result, the designer obtains a model that illustrates the manner in which the final SRIA user interface will appear to any user. In this case study, the UI designer created different screenshots for the representation of different parts of the application, instead of creating a single screenshot.

The first screenshot of the EPM designed by the UI designer, shown in Figure 6.5, contains the login form which will be used in the process of user authentication. The login form, coloured in blue, was created within a *SLCanvas* widget containing two input widgets, i.e., a *SLTextBox* widget and a *SLPasswordBox* widget, and a *SLButton* widget in order to invoke the process of authentication. The context of this screenshot is the *AnonymousUser* navigational class from the ENM. The authentication process is performed by the *login* operation of this navigational class. However, this process and the navigation between screenshots is managed by the Extended Orchestration model.

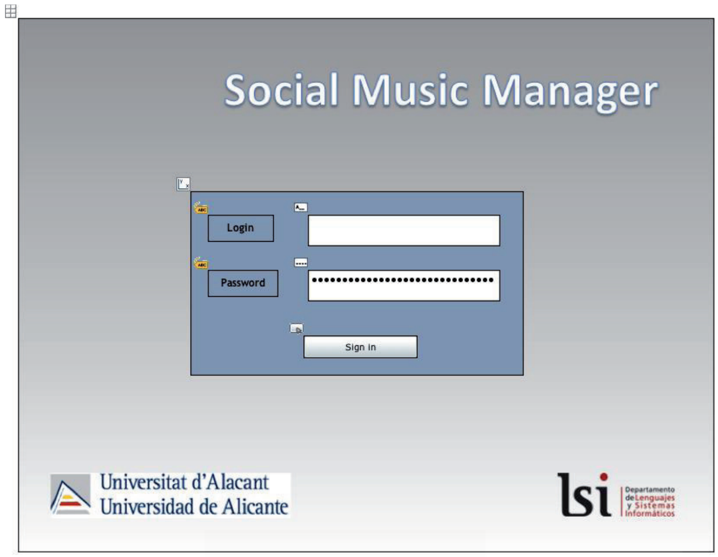


Figure 6.5. Screenshot with the login form of the Social Network Site.

The main screenshot of the EPM, illustrated in Figure 6.6, shows the main social information of the SNS. The widgets in this screenshot can be grouped into three different areas:

a) the UI header, located at the top of the UI, which contains the widgets for the visualisation of the main information about the user (name, email, picture and status) and the widgets (*SLButton* widgets) needed to navigate through the user interface;
b) the information summary, located on the left area of the UI, which contains the widgets that show a summary of the users' friends, albums and tracks using *SLExpander* and *SLListBox* widgets; and
c) the story line, which groups the widgets that show the stories and the comments of the user and their friends, and facilitates their management (creation and deletion).

The schema contained in Figure 6.7 depicts part of the relationships that the UI designer established between this model and the ENM. Specifically, the schema is focused on the visualisation of the user stories. The initial context of the screenshot is the *User* navigational class. The area in which the stories are shown is represented as a *SLStackPanel* widget highlighted with an orange border, and the pattern for the creation of a single story is represented with the inner blue *SLStackPanel* widget. The outer *SLStackPanel* widget is related to the *Story* navigational class in order to indicate the type of the objects visualised and to the *getAllCreatesOfOwner* traversal link in order to indicate the manner in which the objects will be retrieved (in this case, using this automatic link).

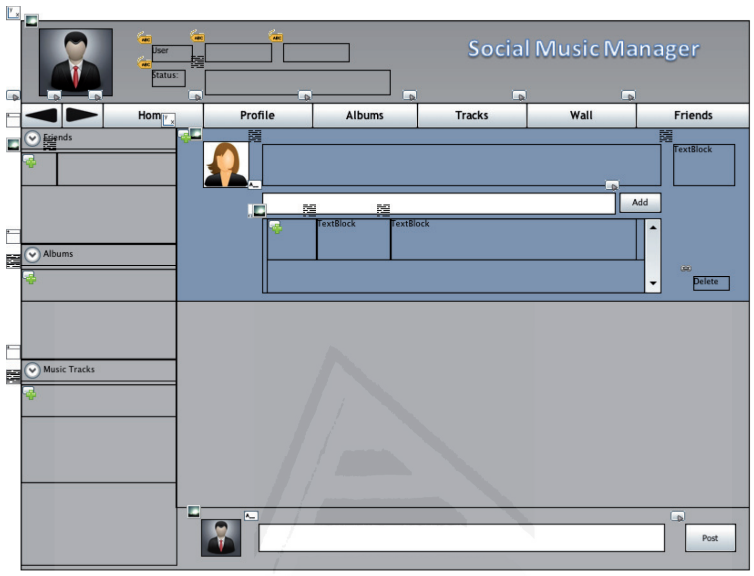


Figure 6.6. Main screenshot of the Social Network Site.

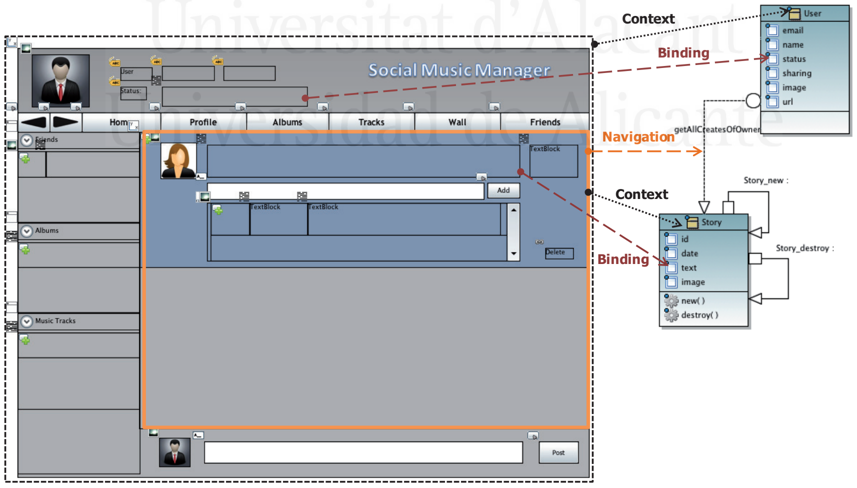


Figure 6.7. Example of the existing relationships between the Extended Presentation Model and the Extended Navigational Model in the main screenshot of the SNS.

The last screenshot of the EPM introduced in this section is illustrated in Figure 6.8, which can be used to visualise information about local or external albums and tracks, depending on which ENM elements is related to. It is supposed to be accessed using the "Albums" *SLButton* widget located at the UI header (even though, the transition between screenshots and the behaviour of this widget will be defined with the EOM).

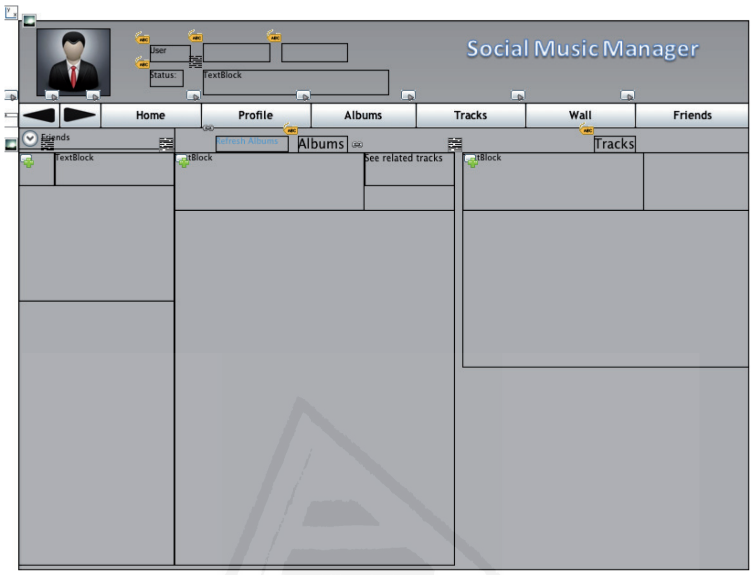


Figure 6.8. Screenshot showing the information of the albums and music tracks of the Social Network Site.

In the same way that the previous screenshot, this is divided in three areas: *a)* the UI header; *b)* the information summary, which, in this case, only represents information about the users' friends; and *c)* the central area, which contains the panels and the widgets for showing the information about the albums and the music tracks. In contrast to the previous screenshot, this screenshot in its central area includes two *SLStackPanel* widgets in order to visualise two lists of objects. The left-hand list will show a list of albums and each of its elements will contain a link to visualise the list of its corresponding tracks.

After completing the design of the screenshot, the designer associated the UI widgets to the external navigational classes of the ENM in order to retrieve and show albums and tracks from MusicBrainz to the users. Figure 6.9 depicts a schema with the main relationships that were established to this aim.

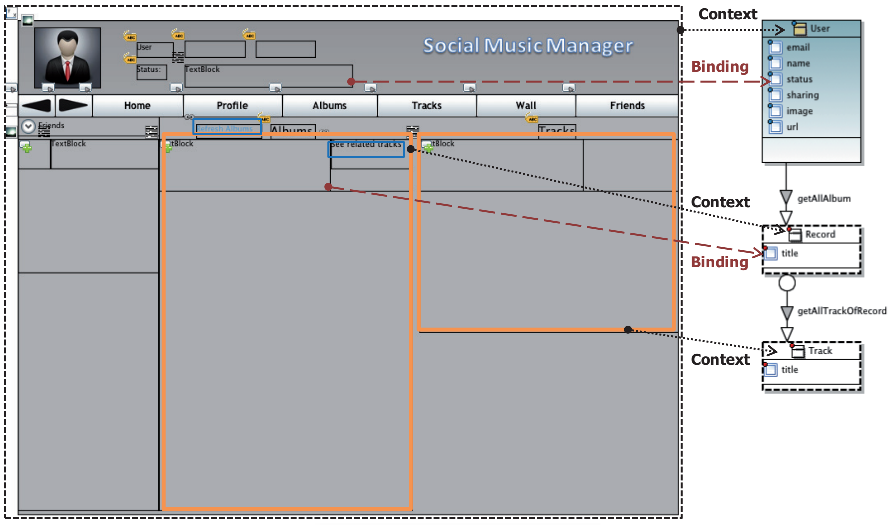


Figure 6.9. Example of the existing relationships between the Extended Presentation Model and the Extended Navigational Model in the "Albums" screenshot of the SNS.

According to the schema, the context of the whole screenshot is the User navigational time again, while the contexts of the two *SLStackPanel* widgets containing the lists of albums and tracks (highlighted with an orange border) are the *Record* and the *Track* external navigational classes, respectively. Since the links that connect these three navigational classes are not automatic, the designer could not establish the Navigation relationship in the containers. In order to populate the two lists of widgets, the designer will define the behaviour of the "Refresh Albums" and "See related tracks" *SLHyperlinkButton* widgets, with a blue border, in the Extended Orchestration model.

6.2 Specifying the Behaviour of the User Interface Components

With the EPM, the UI designer could represent the structure of the UI but, as shown in the last section, there were some behavioural aspects related to the interaction with the server components that could not be defined. Moreover, the EPM cannot manage the specification of client actions, performed by the own widgets, e.g., to hide/show a container, to resize a widget, etc. In the S^{m}4RIA development process, once the design of the UI components is completed, in the subsequent group of tasks, the UI designer focuses on the specification of the behaviour of the UI components using the Extended Orchestration Model, which is a platform-specific model composed by a collection of event-condition-action (ECA) rules that manage the actions performed by the UI components after an event has been triggered on a certain widget.

ECA rules contain three types of elements: a) events, which represent the interaction between users and widgets; b) conditions, which control the flow of actions performed after an event has been triggered; and c) actions, which can be carried out by client (EPM widgets) or server (ENM operations) components. For each widget, there are a fixed collection of events that can be triggered on them and another collection of (client) actions that can be performed, which mainly depend on the technology of the implementation. Figure 6.10 illustrates a diagram with a simple schema of an ECA rule defined over a *SLButton* widget from the EPM, i.e., a button of a screenshot of the UI.

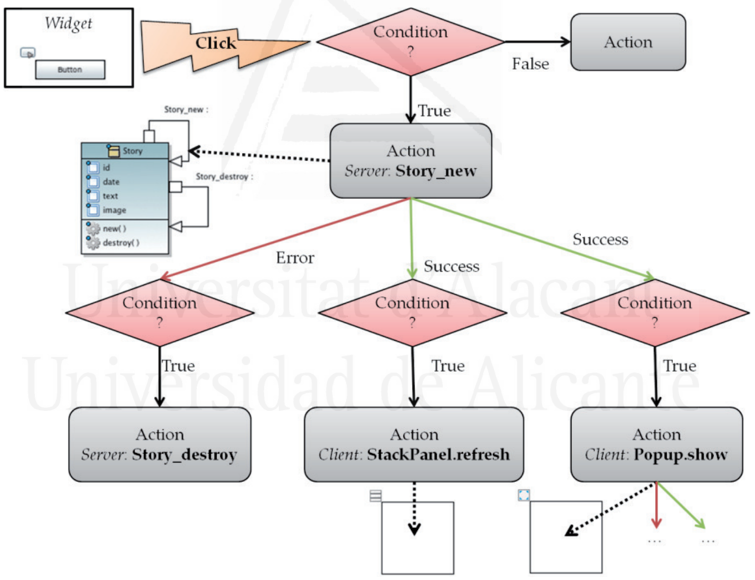


Figure 6.10. Representation of an Event-Condition-Action rule.

In this case, after a user clicks this widget, thus triggering the *Click* event, a condition will be checked. This logic condition can check the state of other UI widgets and, depending on the result, i.e., *true* or *false* values, it would be possible to perform different actions. It is supposed that the *Story_new* action is performed if the condition is satisfied. This server action invokes the *Story_new* service link of the ENM, which creates a new story (*Story* object) from a set of parameters. The parameters for this action can be obtained from the contexts in which the widget is contained or from input widgets such as *SLTextBox* widgets. In the EOM, an action only has two possible return values: *success* or *error*, which indicate whether the action was completed or there was an error during the process.

As can be appreciated, the EOM facilitates the concatenation of actions creating sequences. After an action has been completed and depending on its return value, it is possible to invoke different actions always if their corresponding conditions are satisfied. According to the figure, in case of error, the client intends to delete the object, using the *Story_destroy* client action, depending on the type of error (checked by the condition). In case of success, the rule invokes two client actions: *a)* it refreshes the content of the stack panel (*SLStackPanel* widget) that contains the list of Story objects (*StackPanel.refresh* action); and *b)* it shows a pop-up window (*SLPopup* widget) indicating that the action finished successfully (*Popup.show* action). In this second level of actions, the designer can also employ the actual result of the first operation as a parameter using the keyword "Result". If the service link invoked by the action contains an alternative target getter, the designer can use the keyword "ResultAlt" in order to access the data resulting of that operation. Following the described pattern, the designer can continue the specification of the ECA rules according to the requirements of the application.

In S^{m}4RIA, the process of design of the EOM is similar to the EPM's (please, check the SPEM2 diagram in Figure 4.2, page 70). Using the *Nav&Pres2Orch* M2M transformation and the ENM and EPM as inputs, it is possible to generate a mock-up of the EOM which infers the behaviour of the UI during the invocation of the predefined navigational operations (i.e., those whose type not *custom)*. However, given the variability and complexity of the user interface, the result of the transformation might not help the designer when addressing UIs created from scratch. It is recommended that the *Nav2Pres* and *Nav&Pres2Orch* transformations are used together in those cases in which the variability of the user interface is reduced, i.e., in the design of UIs for administrators. This section will not contemplate this scenario and is focused on the manual specification of the UI behaviour using the ECA rules of the Extended Orchestration Model. The process of the specification of ECA rules does not follow any pattern and ECA rules are contained in a plain unordered list. Therefore, in the process of creating the EOM, the designer specifies ECA rules for those widgets and events they consider relevant (according to the stakeholders' requirements) only ensuring that the rule structure is correct.

6.2.1 Extended Orchestration Metamodel: Concrete and Abstract Syntaxes

The structure of the ECA rules is formally described in the Extended Orchestration metamodel, which the EOM is conformant to. This metamodel captures the semantics of all the elements of the ECA rules and provides a graphical representation based on the UML state diagram. Figure 6.11 depicts a schema of the main elements of this metamodel. Those elements with the *EPM* and *navigationalView* prefixes are imported from the Extended Presentation metamodel and the Extended Navigational metamodel, respectively.

The main element of this metamodel is the Widget element, which is imported from the EPM, since the purpose of the EOM is to define the behaviour of the structural elements of the EPM. The elements of the metamodel can be classified into two groups:

a) Elements extending the concept of widget and screenshot (from the EPM), which define the concept of widget event, widget action and their properties. This group contains the following elements: *WEvent*, *WMethod*, *WEventProperty* and *WMethodProperty*. These are necessary for the specification of ECA rules.
b) Elements for the specification of ECA rules, which define the elements required for the association of a certain event with the actions performed. The metaclass that represents the whole rule is the *EventCall* metaclass.

In this metamodel, only the second group of elements, including the *Widget* metaclass, has a graphical representation, which, as mentioned before, is imported from the UML state diagram. This might seem a contradiction taking into consideration the representation of the ECA rule presented in the previous section, which showed the rule as a kind of sequence of actions. However, given that RIA widgets are stateful elements, the notation of the UML state diagram is more appropriate than the one of UML sequence diagram. Table 6.3 introduces the elements of the Extended Orchestration metamodel including the description of each of element and its graphical notation.

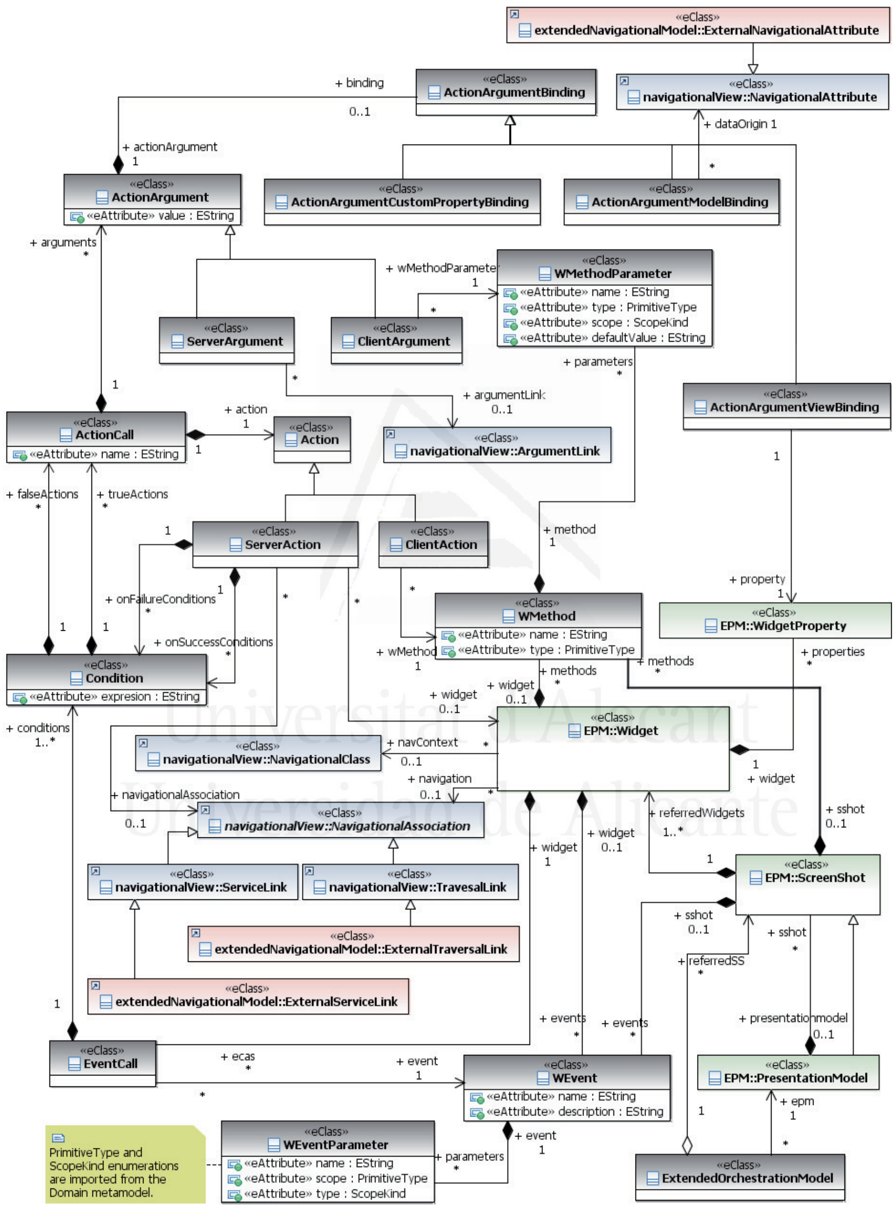
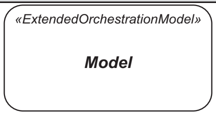
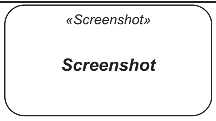
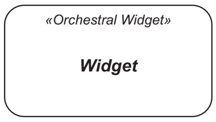
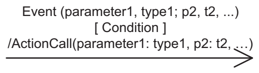

extendedNavigationalModel::ExternalNavigationalAttribute
ActionArgumentBinding
navigationalView::NavigationalAttribute
ActionArgument
ActionArgumentCustomPropertyBinding
ActionArgumentModelBinding
WMethodParameter
ServerArgument
ClientArgument
ActionArgumentViewBinding
ActionCall
Action
navigationalView::ArgumentLink
ServerAction
ClientAction
WMethod
EPM::WidgetProperty
Condition
EPM::Widget
navigationalView::NavigationalClass
navigationalView::NavigationalAssociation
navigationalView::ServiceLink
navigationalView::TravesalLink
EPM::ScreenShot
extendedNavigationalModel::ExternalTraversalLink
extendedNavigationalModel::ExternalServiceLink
EventCall
WEvent
EPM::PresentationModel
WEventParameter
ExtendedOrchestrationModel
PrimitiveType and ScopeKind enumerations are imported from the Domain metamodel.

Figure 6.11. Schema of the Extended Orchestration Metamodel.

Table 6.3. Description of the main elements of the Extended Orchestration metamodel.

Metaelement	Description	Graphical notation
ExtendedOrchestra-tionModel	This metaclass represents the whole EOM model and contains all the elements of the ECA rules.	
Screenshot	This metaclass, imported from the EPM, represents the screenshots of the UI.	
Widget	This metaclass, imported from the EPM, represents the structural elements of the UI. It is the central metaclass for the EPM and the EOM.	
WidgetProperty	It represents a property of a widget. Each subtype of widget has different properties according to its characteristics, e.g., the textbox widget has the text property, which cannot be present in the stack panel.	*No graphical representation.*
WEvent	It represents an event that can be triggered on a certain widget.	*No graphical representation.*
WEventParameter	This metaclass represents a parameter of the event. For the management of certain types of interaction, such as the modification of a textbox widget, it	*No graphical representation.*
WMethod	It represents an action that can be performed on a certain widget.	*No graphical representation.*
WMethodParameter	This metaclass represents the parameters of an action.	*No graphical representation.*
EventCall	This metaclass captures the semantics of an ECA rule	 Event (parameter1, type1; p2, t2, ...) [Condition] /ActionCall(parameter1: type1, p2: t2, ...)
Condition	It represents a condition of an ECA rule. This condition must be specified in OCL.	*See the EventCall metaclass.*
Action	It represents an action of an ECA rule.	*No graphical representation.*
ClientAction	It represents an action of an ECA rule performed by a widget of the UI, i.e., the invocation of a *WMethod* element. Each object of this	*No graphical representation.*

Metaelement	Description	Graphical notation
	metaclass must be associated to an object of the *WMethod* metaclass.	
ServerAction	It represents an action of an ECA rule performed by a service of the SRIA server, i.e., the invocation of a navigational operation of the ENM. The objects of this metaclass are associated to the object of the *WMethod* metaclass.	*No graphical representation.*
ActionCall	This metaclass represents an invocation of an action within an ECA rule.	*See the EventCall metaclass.*
ActionParameter	It represents the parameters of the invocation of an *ActionCall* element. Depending on the type of action invoked, the parameters of the *ActionCall* element must correspond to the parameters of the *WMethod* element or to the arguments of the navigational operation.	*See the EventCall metaclass.*

Table 6.4 presents some of the most relevant OCL constraints of the Extended Orchestration metamodel, which were reused from the OOH4RIA Orchestration metamodel.

Table 6.4. Summary of the OCL constraints defined over the Extended Orchestration metamodel.

Name	Description / OCL Contraint
ManageOwnEvents	An *EventCall* element must only manage *Event* elements of its own *Widget* element.

	`Context EventCall`
	`inv:`
	`self.widget.events->include(self.event)`

CorrectBinding	*ActionArgumentModelBinding* elements must be related to the *NavigationalAttribute* elements associated to the context of the *Widget* element that triggers the event and manages it.

	`Context EventCall`
	`inv:`
	`-- Ignoring the loop Condition-ActionCall-`
	`-- Action-ServerAction.`
	`self.conditions->trueActions->union(`
	`self.conditions->falseActions)->arguments->`
	`binding->select(b |`

Name	Description / OCL Contraint
	`b.oclIsTypeOf(ActionArgumentModelBinding))-> forAll(b | self.widget.navContext.navAttributes-> union(self.widget.containers()-> navAttributes)->include(b))`
CorrectParametersInActionCall	*ActionCall* elements must assign a value to all the parameters of the *Action* elements performed (either *ClientAction* or *ServerAction* elements).
	`Context ActionCall inv: -- For Server actions (self.action.oclIsTypeOf(ServerAction)` **`and`** `self.action.oclAsType(ServerAction). navigationalAssociation.oclIsTypeOf(ServiceLink)` **`and`** `self.arguments->forAll(a | a.oclIsTypeOf( ServerArgument))` **`and`** `self.arguments->includesAll( self.action. oclAsType(ServerAction).navigationalAssociation.oclAsType(ServiceLink).argumentLink))` **`or`** `-- For client actions (self.action.oclIsTypeOf(ClientAction)` **`and`** `self.arguments->forAll(a | a.oclIsTypeOf( ClientArgument))` **`and`** `self.arguments.oclAsType(Set(ClientArgument))-> wMethodParameter->includesAll( self.action.oclAsType(ClientAction).wMethod.parameters)`

In order to facilitate the understanding of the components of this metamodel, the next section will introduce the EOM for the case study of the development of the SNS and will show a set of the most frequently used ECA rules.

6.2.2 An Extended Orchestration Model for the Social Network Site

Once the designer completes the EPM of the SNS case study, as explained in the introduction of Section 6.2, they specify the behaviour of the UI elements with a set of ECA rules creating a new Extended Orchestration model.

Figure 6.12 depicts a fragment of the EOM resulting of this process, which represents a subset of the ECA rules defined over the UI elements. In this diagram, all the elements (from the model element to the orchestral widgets) remain idle waiting until an event is triggered on any of them. The state diagram shown in the figure contains examples of the three main scenarios in which the designer might need to specify an ECA rule:

a) *Screenshot navigation.* The designer can specify the navigation between screenshots when designing multi-page (S)RIAs.
b) *Service invocation.* It is normally necessary to access data from the SRIA server or to invoke an operation by means of the SRIA Web services.
c) *UI modification.* Another frequent scenario is the modification of the UI interface, i.e., the client components need to perform operations that affect the visual appearance of the UI.

The first example involves the Screenshot elements of the EOM. In this case, the designer specified the behaviour of three screenshots, i.e., *Login*, *Main* and *PersonalInformation*, corresponding to the ones designed in Section 6.1.2 (page 126), and the navigation between them. According to the model, the initial screenshot of the application is the *Login Screenshot* element (the EPM does not specify a sequence of screenshots). This model does not represent the behaviour of the widgets contained in the *Login* screenshot but manages the *onSubmitCompleted* event, which indicates that an action has been completed and, particularly, the login action, which authenticates the users in the SRIA. This event contains two parameters: the widget which the event was triggered from (which is a parameter included in all the events), and the result of the action performed. The condition of this ECA rule checks whether the result is valid and, if so, the application navigates from *Login* to *Main* and establishes the result of the *login* navigational operation as the context of the *Main* screenshot (i.e., a User navigational object according to the ENM of this application).

The designer also defined the *log-out* process by managing the *onClick* event on the *Main* or *PersonalInformation Screenshot* elements. In this case, the condition of these ECA rules check if the event was focused on the *SignOut SLButton* element and invokes the *signOut* server action.

In order to exemplify the second scenario, the designer defined an ECA rule that manages the creation of a new story from the user data and subsequently refreshes the list of stories shown in the user wall (*SLStackPanel1* orchestral widget).

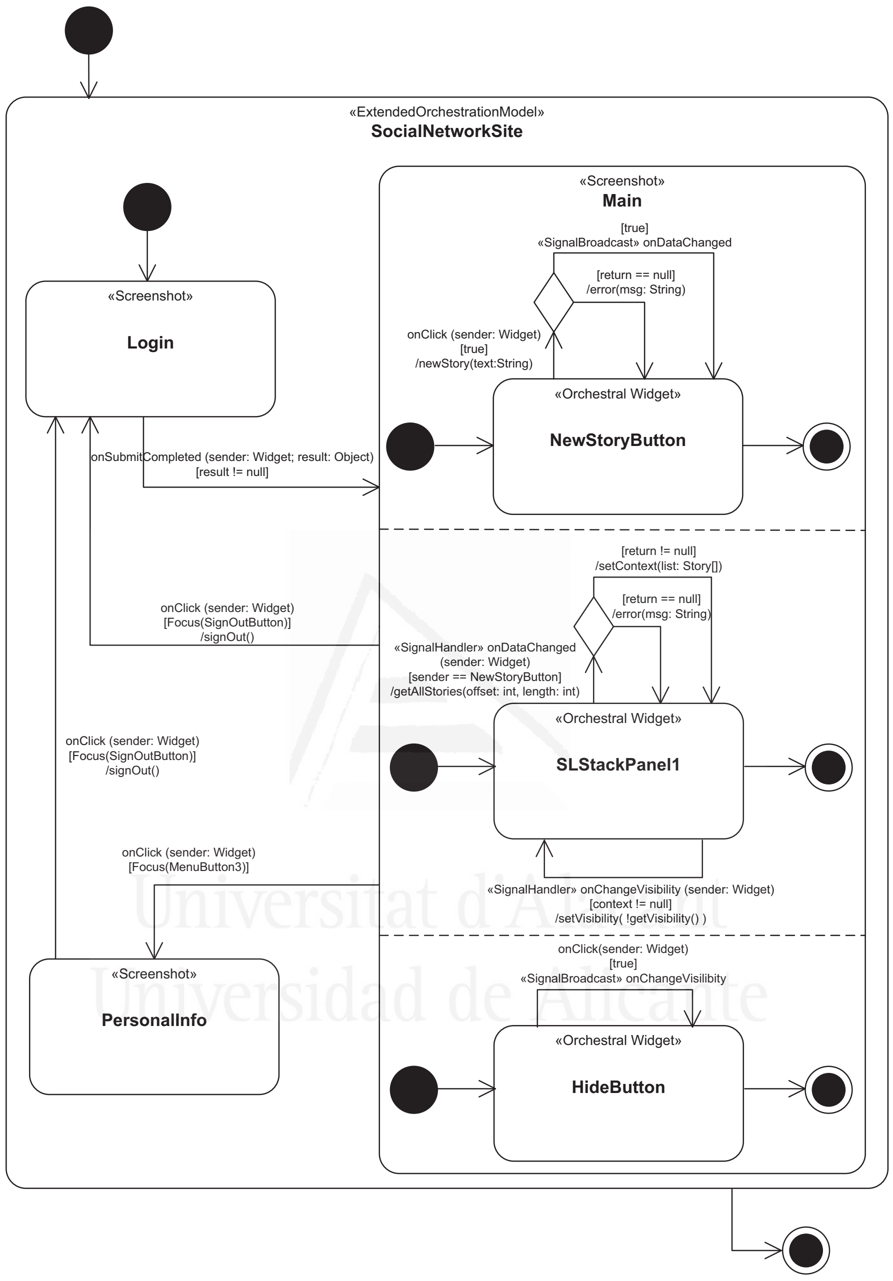


Figure 6.12. Fragment of the Extended Orchestration Model for the SNS case study.

This ECA rule starts when the user triggers the *onClick* event of the *NewStoryButton SLButton* widget, which is part of the "New story" form. When users introduce the body of their story in the given text box and click on this button, the application should create a new story by means of the *newStory* server action. The diagram only shows one of the parameters of the newStory server action, i.e., the text of the story, but it is also necessary to indicate the ID of the user who created it and the date of creation. The assignation of value to these parameters is also performed in the EOM but does not have any graphical representation. The possible values of these parameters can be classified into three groups:

- *Context values*. These values are obtained by associating the properties (navigational attributes) of one of the contexts (navigational classes) in which the widget is contained to the parameter. For instance, the user identifier is obtained from the *User* navigational class, which is the context of the screenshot.
- *UI values*. These values are obtained by associating the properties of the widgets contained in the screenshot to the parameter. For instance, the text of the story is obtained from the text property of the *SLTextBox* widget, part of the "New story" form.
- *Constant values*. The third type of values is directly introduced by the designer. Constant values can be specified in OCL, when possible, or in the language of the implementation technology, in this case, C#. For instance, in order to obtain the time and date when the story is created, the designer could use the following C# expression: "DateTime.Now".

The new story is shown in the stack panel located just above the input form (SLStackPanel1 Orchestral widget). If the server action is successfully completed (`return!=null`), the *NewStoryButton* element broadcasts a *onDataChanged* signal in order to notify the rest of the widgets that the data has been modified. If there were an error in the process, the application should show an error message (using the *error* client action) using the default output method, which depends on the technology (in this case, a message box will be used).

The *onDataChanged* signal previously emitted is managed by the *SLStackPanel1* widget, which retrieves an up-to-date list of stories from the SRIA server using the *getAllStories* server action. If the new list is successfully retrieved the *setContext* client action updates the context of the stack panel and, as a consequence, the content is redrawn showing the new story.

The third, and final, scenario is the modification of the appearance of the UI using an ECA rule. In the example, the *HideButton* SLButton element manages the visibility of the list of stories in the user wall. The designer created an ECA rule that manages this behaviour when the *onClick* event of the *HideButton* widget is triggered. Subsequently, the widget emits the *onChangeVisibility* signal, which is captured and managed by SLStackPanel1 widget. This orchestral widget checks whether it is empty (or its context is null), and if it contains elements, it modifies its *visibility* property, showing or hiding the contents with the *setVisibility* and *getVisibility* client actions.

Using the solutions proposed for these three scenarios, it is possible to define the behaviour of the rest of the UI widgets.

6.3 Generating an User-Oriented, Ontology-Based Representation of the User Interface

Once the Extended Presentation and Orchestration models have been completed, using the *Pres&Orch2Visu* model-to-model transformation, the transformation engine can automatically generate the Visualisation Ontology Model (VOM) from the previous models. This model gathers and combines information from the abstract presentation and orchestration concepts, obtaining as a result an abstract representation of the UI elements and their behaviour from the perspective of the final users, which avoids implementation details. The transformation merges information about the structure and behaviour of UIs hiding those aspects that can affect the security of the application, which should only be known by the designer.

6.3.1 The OntoVisu Metamodel

The elements of the VOM are conformant to a new ODM-based metamodel was created, called OntoVisu, which defines mechanisms to describe the UI from the users' perspective combining the structural elements of a SRIA UI with the behavioural aspects. Figure 6.13 depicts a schema with the main elements of the abstract syntax of this metamodel and the relationships to elements of other metamodels, i.e., Extended Navigational metamodel (whose elements are coloured in red), Extended Presentation and Orchestration metamodels (coloured in blue and green, respectively) and the *OWLBase* metamodel (coloured in orange). The main element of OntoVisu is the *VisualElement* metaclass, which abstracts the semantics of any elements involved in the visualisation of the UI. The elements of this metaclass can be classified as structural or behavioural elements, which are related to the elements of the EPM or the EOM, respectively.

The elements of the VOM are also associated to the elements of the ENM by means of three different patterns (in a similar manner that the EPM and the EOM):

a. *ElementContainer* components can be linked to a navigational element;
b. Action elements can be associated to a service link from the ENM;
c. The specification of the action parameters, their types and results are optional and linked to the arguments of the navigational operations.

The hierarchy of structural elements is similar to the EPM's, since, in this case, the perspective between the designer and the users does not differ considerably. The main issue is the different perspective of the behaviour of the UI that users and developers have. While designers can specify the exact behaviour of the application with ECA rules, users should only perceive a biased image of what is actually happening, mainly due to security issues.

The *OntoVisu* metamodel captures the possible interactions between users and the UI elements and the actions they finally notice in the user interface. The transformation proposed generates a model representing all the information that is possible to represent with this metamodel. Depending on the security level of the target application, the transformation should be adjusted in order to show different levels of information.

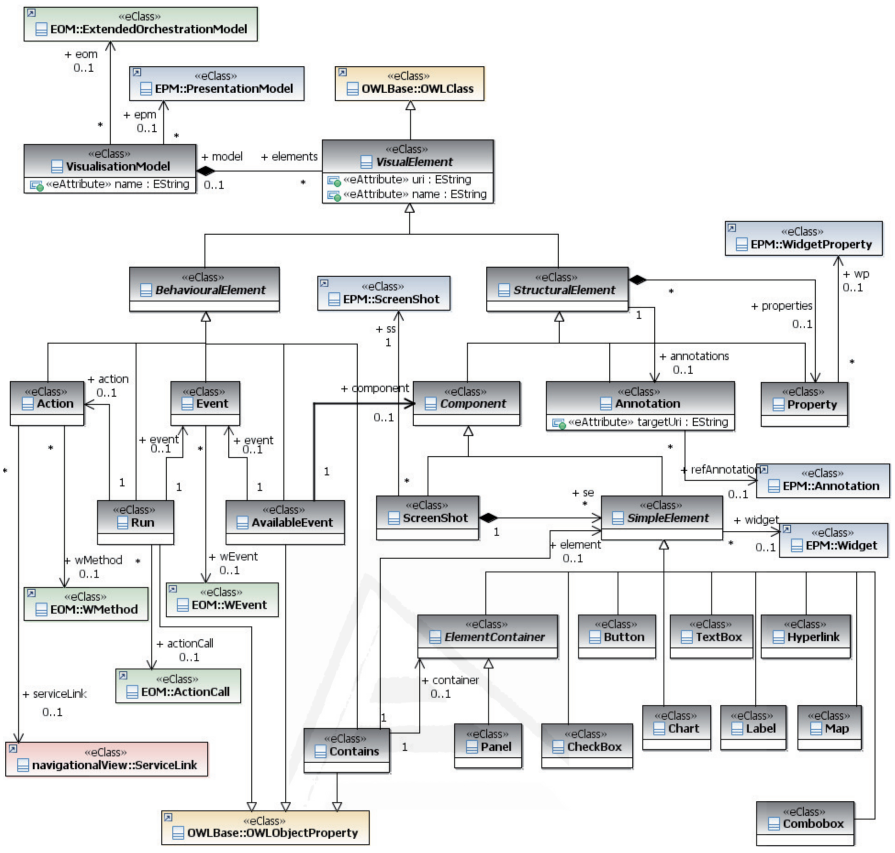


Figure 6.13. Main elements of the OntoVisu metamodel (abstract syntax).

Table 6.5 contains the description of the main elements of the metamodel and their graphical notation. This table only describes the most general elements and the ones with a graphical representation.

Table 6.5. Description of the main elements of the OntoVisu metamodel.

Metaelement	Description	Graphical Representation
VisualisationModel	This metaclass represents the whole model. A *VisualisationModel* element is related to a *PresentationModel* element and a *ExtendedOrchestrationModel* element, which is created from.	*No graphical representation.*
VisualElement	This is an abstract metaclass that represents all the elements contained in a Visualisation element.	*No graphical representation.*

Metaelement	Description	Graphical Representation
Screenshot	It represents a screenshot of the UI containing a set of Component elements in a certain state. There are two graphical notations depending on whether the screenshot is the initial one or not.	👁 👁
StructuralElement	Abstract metaclass that represents any structural component of a *Visualisation* element (e.g., screenshots, panels and widgets). Visual elements can be grouped into three categories: components, properties and annotations.	*No graphical representation.*
SimpleElement	This metaclass is a specialisation of the *Component* metaclass and represents a widget with a single functionality which cannot contain other widgets.	🧩
ElementContainer	This specialisation of the *Component* metaclass represents complex widgets that can contain other ones.	🧩
BehaviouralElement	Abstract metaclass that represents the elements involved in the interaction between users and the structural elements of the interface.	*No graphical representation.*
Event	This metaclass represents the events that users can trigger on a certain widget. The elements of this metaclass can be related to Action elements specifying which part of the SRIA performs the action (server or client).	⚡
Action	The Action metaclass represents those actions that can be perfomed by a Widget element. Actions must be associated to, at least, one event.	⚙
Annotation	This metaclass represents a static semantic annotation over a certain widget.	URI reference

Table 6.6 introduces some of the main OCL constraints defined over the OntoVisu metamodel in order to maintain the consistency of the information stored in the model.

Table 6.6. Summary of the OCL constraints defined over the OntoVisu metamodel.

Name	Description / OCL Contraint

DifferentComponentName	Two *Component* elements cannot share the same name in the same model.
	`context VisualisationModel`
	`inv:`
	`        self.elements->select(e |`
	`        e.oclIsTypeOf(Component))`
	`        ->forAll(c | self.elements->select(e2 |`
	`        e2.oclIsTypeOf(Component) and e2.name =`
	`        c.name)->size() = 1)`

CompleteContainsRelationship	All the *Contains* elements must relate an *ElementContainer* element to a *SimpleElement* element. (This OCL constraint could be replicated for any of the subclasses of the *OWLObjectProperty* metaclass)
	`context VisualisationModel`
	`inv:`
	`        self.rels->select(r |`
	`        r.oclIsTypeOf(Contains))->forAll(c |`
	`        c.oclAsType(Contains).element <> null and`
	`        c.oclAsType(Contains).container <> null)`

SameNameSimpleElementWidget	The name of the *SimpleElement* must be equal to the name of the widget from the EPM to which it is associated. (This OCL constraint could be replicated for any of the elements with a relationship to elements of other models)
	`Context SimpleElement`
	`inv:`
	`        self.name = self.widget.name`

SameElementsSameScreenshot	A ScreenShot element must contain the *SimpleElement* elements created from the widgets associated to the screenshot to which is linked (from the EPM).
	`Context ScreenShot`
	`inv:`
	`        self.ss.referredWidgets->`
	`        includesAll(self.se->widget)`

OnlySimpleElementsHaveWidgetProperties	Only the Property elements associated to SimpleElement elements can be associated to widget properties of the EPM.
	`Context StructuralElement`
	`inv:`
	`        (self.oclIsTypeOf( SimpleElement ) and`
	`        self.properties->wp->size() <> 0) or not`
	`        self.oclIsTypeOf( SimpleElement )`

SameWidgetSameProperties	The widget properties associated to the Property elements of a SimpleElement element must have been obtained from the

Name	Description / OCL Contraint
	properties of the widget (of the EPM) to which it is associated.

	`Context SimpleElement`
	`inv:`
	`self.widget.properties->`
	`includesAll(self.properties->wp)`

Using these elements the *Pres&Orch2Visu* M2M transformation rule creates the VOM from the information contained in the model (see Section 7.3.3, page 199). The use of a graphical notation for a model generated automatically (and should not be updated according to the S^{m}4RIA process) is motivated by the actual necessity of offering a human-oriented visualisation in some cases when the designer needs to manually constrain the output model (as mentioned before). The use of a transformation rule also ensures that the OCL constraints are satisfied from the creation of the model

The next section describes the model resulting from the transformation of the Extended Presentation and Orchestration models of the SNS case study. The transformation process proposed will be described in Section 7.3.3 (page 199) with the rest of the model-2-model transformations defined in the methodology.

6.3.2 THE VISUALISATION ONTOLOGY MODEL OF THE SOCIAL NETWORK SITE

After completing the Extended Orchestration Model of the social network site, the designer invokes the Pres&Orch2Visu transformation process obtaining the VOM. Figure 6.14 illustrates a diagram showing part of the VOM of the case study corresponding to the EPM and EOM introduced in Section 5.2.2 and Section 5.3.2. The figure illustrates a schema containing those visual elements involved in the process of user authentication and the creation of a new story.

The initial element of this model is the Login *ScreenShot* element, illustrated as an open eye, which represents the first screenshot of the SRIA (see Figure 6.5). Each type of visual element (illustrated as jigsaw pieces) is an instantiation of the subclasses of the VisualElement metaclass, such as Grid or TextBox. These model elements are related to a set of properties, represented as model annotations, for the representation of the aesthetic characteristics of each visual element, such as the size, the relative position or colour. For instance, the *size* property of this screenshot is set to *"1024x768"*.

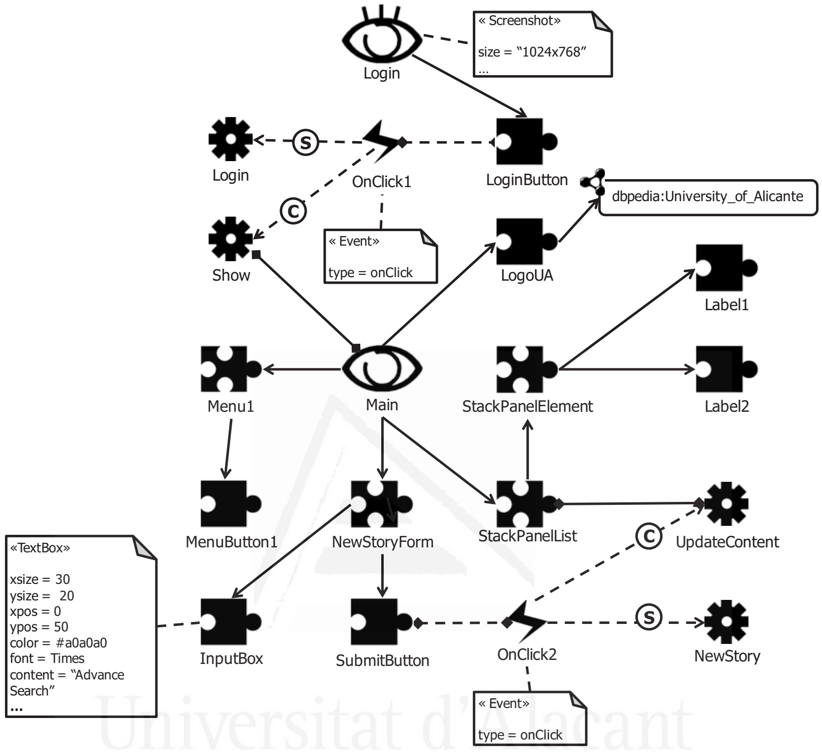


Figure 6.14. Fragment of the Visualisation Ontology Model of the SRIA MediaPlayer.

According to the diagram (which represents a subset of all the elements of the screenshot due to spacing constraints), the *Login* screenshot contains the *LoginButton* Button element. The diagram represents the process of user authentication, which is performed after the *OnClick1* event is triggered on that component and involves the invocation of the Login and Show actions. The second screenshot depicted in the model is the Main screenshot (see Figure 6.6), which contains four components:

a. *Menu1*, which represent the menu panel located at the header of the screenshot;
b. *LogoUA*, which represents an image with an ontology-based annotation.
c. *StackPanelList*, which represents the component used for the creation of the list of stories. The *StackPanelElement* component

represents the component used as a pattern for the creation of the stories within the list. The information about the stories is located within the *Label1* and *Label* 2 components.

d. *NewStoryForm*, which represents the form designed for the creation of stories. This form contains an input component, called *InputBox* and conformant to the *TextBox* metaclass, and the *SubmitButton* component that manages the process of creation by means of the *OnClick2* event.

As just mentioned, in this second screenshot, the *LogoUA* component represents an image associated to an ontology-based association, mainly characterised by a URI. This image, which illustrates the institutional logo of the University of Alicante, is associated to the resource identified by *http://dbpedia.org/resource/University_of_Alicante*, which refers to the *University_of_Alicante* entity of the DBpedia Linked Data repository. This type of annotations can be used to retrieve extra information associated to the content of the image.

This is the last model of the S^{m}4RIA process for the SNS case study. With the information contained in the collection of models described, the model-2-text transformation engines automatically generate the software modules of the application.

6.4 Conclusions

This chapter has explained the tasks and models of the second activity of the S^{m}4RIA process, i.e., the design of the UI, in which the UI designer specifies all the elements needed for the generation of the SRIA client for human users taking into consideration the stakeholders' requirements and the models of the previous activities.

The tasks in this activity can be grouped into two main sequences, depending on the aspects of the user interface they address: *a)* the design of the UI structure and visual appearance using the Extended Presentation Model or *b)* the design of the behaviour of the UI structural components with the Extended Orchestration Model. Although, in some cases, these models can be created using a model-2-model transformation process, this chapter described the process of manual specification of the models, which can be employed in the development of any case study.

Both models are platform-specific and are adapted to the development of Silverlight (S)RIAs. In order to facilitate the adaptation of these models to any other technology, the Extended Presentation and Extended Orchestration metamodels contain groups of abstract elements, independent from the platform, which create the most general elements that are specialised by the platform-dependent elements and establish the main relationships between these models and the ones explained in the previous chapter.

The information contained in both models was subsequently merged and transformed into the elements of the Visualisation Ontology Model, which provides an overview of the UI from the users' viewpoint. This process is performed by means of a M2M transformation.

In order to clarify the concepts described and assess the proposal in a qualitative manner, the chapter presented the design of the EPM, the EOM and the VOM for a social network site, following the example of the previous chapter.

Chapter 7. GENERATING THE SOFTWARE MODULES OF A SEMANTIC RIA THROUGH MODEL TRANSFORMATIONS

In the first two activities of the S^m4RIA process, designers with the help of a set of model-to-model transformation capture and represent all the design information of the SRIA. The S^m4RIA models contain the details required for the development of a SRIA in an automatic manner. Using this information, in the last S^m4RIA activity, the model-2-text transformation engines obtain the components of the final SRIA by means of a set of model-to-text transformations.

Model-to-text transformations define a set of rules that transform the objects of a collection of input models into the final code of the application. These transformations can generate any detail of the application taking into consideration the needs of the target application (technology, architecture, platform, etc.) and the ones of the developers (development environment, coding standard, etc.) Therefore, they are adapted to each specific case and should be modified whenever the needs (or non-functional requirements) change. The first step before defining the transformation rules is to specify the detailed architecture of the target application, which facilitates the modularisation of the rules and improves their maintainability. For each of the software components, the designer can subsequently specify a collection of rules that fits the requirements.

This chapter introduces the transformation processes required to generate the software modules contained in SRIAs (see Figure 3.3, page

49). Firstly, it will introduce the model-2-text transformations that generate the code of the SRIA server and, subsequently, the model-2-model transformations that can speed up the development process by creating mock-ups of some of the S^{m}4RIA models. While M2M transformations are independent from the implementation, M2M depend on the implementation of the SRIA proposed. Therefore this chapter introduces the general architecture proposed for a SRIA based on a set of well-known architectural design patterns. Subsequently, it presents the adaptation of the general architecture to the development of Silverlight SRIAs and the M2T transformations that generate each software component. Finally, these transformations are implemented in the *S^{m}4RIA extension for OIDE*, which is an extension for the OIDE CASE tool that implements the S^{m}4RIA model editors and code generators.

In this way, this chapter completes the S^{m}4RIA proposal, by describing the last of its activities and the implementation of the models and transformations in a case tool. The example of SRIA architecture introduced could be used as a reference for future implementations.

7.1 THE ARCHITECTURE OF A SEMANTIC RIA

The SRIA structure of the SRIA described in Section 3.3 contains the main software modules that a SRIA must implement. However, the description does not contain the details required for the implementation of a prototype. During the development of the SRIA case studies, different architectures were tested for the Web application. Using the experience gained from that process, this section presents one of the possible SRIA architectures, which specify the components of the application and their organisation. This proposal is based on a set of mature architectural design patterns, which facilitate the design of the components and, at the same time, qualitatively validate the resulting application. Architectural patterns do not ensure that the resulting application is well developed but they represent solutions proved to be successful in several scenarios.

Before starting the description of the architecture proposed, Figure 7.1 illustrates the schema shown in Figure 3.3 (page 49) with a different colour scheme.

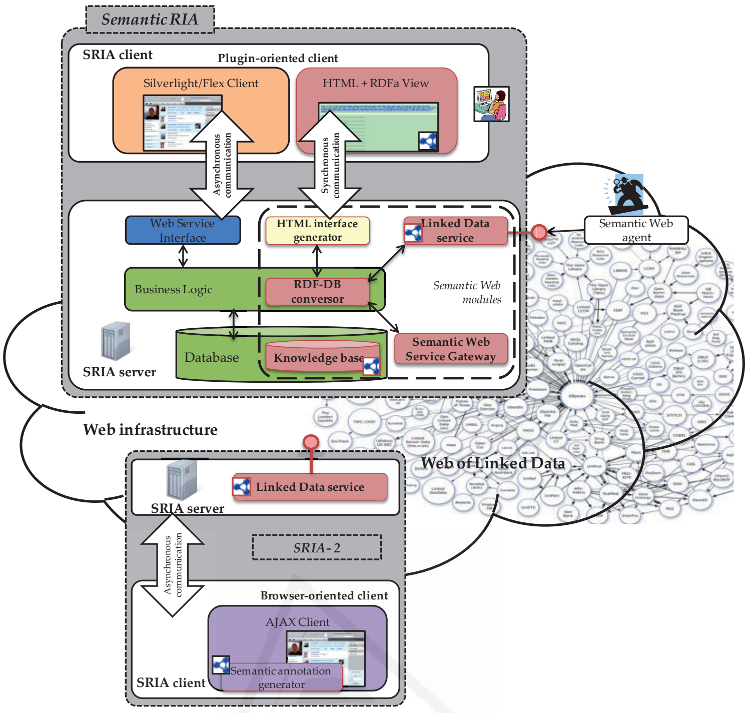


Figure 7.1. Structure of a Semantic RIA.

Taking this diagram as a reference, the following subsections will describe the components of each SRIA module, starting from the description of the architecture of the SRIA server. The system architecture will be specified as a collection of WebSA Configuration models (Meliá, 2007), which extend the UML component models with a new profile for the specification of architectural patterns and the interaction between the components and users or legacy systems.

7.1.1 The Architecture of the SRIA Server

Figure 7.2 shows the WebSA Configuration model of the architecture proposed for a SRIA server. In this case, the diagram introduces a general architecture, which avoids technological details and could be therefore applied to any case study. The colour scheme chosen corresponds to the one of Figure 7.1 and the component names are coherent with the names of the SRIA modules.

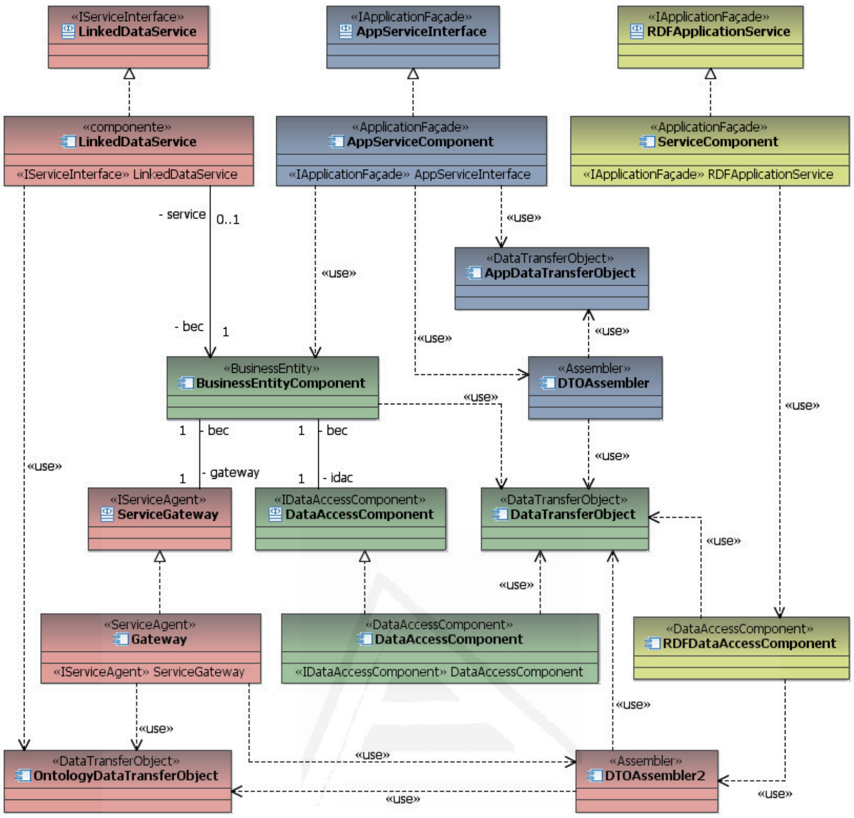

«IServiceInterface» LinkedDataService
«IApplicationFaçade» AppServiceInterface
«IApplicationFaçade» RDFApplicationService
«componente» LinkedDataService
«IServiceInterface» LinkedDataService
«ApplicationFaçade» AppServiceComponent
«IApplicationFaçade» AppServiceInterface
«ApplicationFaçade» ServiceComponent
«IApplicationFaçade» RDFApplicationService
«DataTransferObject» AppDataTransferObject
«BusinessEntity» BusinessEntityComponent
«Assembler» DTOAssembler
«IServiceAgent» ServiceGateway
«IDataAccessComponent» DataAccessComponent
«DataTransferObject» DataTransferObject
«ServiceAgent» Gateway
«IServiceAgent» ServiceGateway
«DataAccessComponent» DataAccessComponent
«IDataAccessComponent» DataAccessComponent
«DataAccessComponent» RDFDataAccessComponent
«DataTransferObject» OntologyDataTransferObject
«Assembler» DTOAssembler2
- service 0..1
- bec 1
- gateway 1
- idac 1
«use»

Figure 7.2. Configuration model for the general architecture of the SRIA server.

Table 7.1 shows the components included in all the SRIA server modules, which links the SRIA structure to the components of the Configuration model.

Table 7.1. Mapping between the SRIA server modules and architectural components.

SRIA server modules	Architectural components
Database, knowledge base	DataAccessComponent (interface and component), DataTransferObject
Semantic Web Gateway	ServiceGateway, Gateway, OntologyDataTransferObject, DTOAssembler2
Business Logic	BusinessEntityComponent
Web Service Interface	AppServiceInterface, AppServiceComponent, AppDataTransferObject, DTOAssembler

SRIA server modules	Architectural components
HTML interface generator	RDFApplicationService, ServiceComponent, RDFDataAccessComponent
Linked Data Service	LinkedDataService (interface and component)

As mentioned before, WebSA Configuration models allow designers to specify the architectural components, interfaces and the relationship between them. The WebSA profile includes a collection of stereotypes that represent well-known architectural patterns (obtained from different authors) that can be applied on the components. In this case, the following patterns were employed during the design of the server architecture (described by Meliá, 2007):

- **Data Access Object** (Alur et al., 2003). It defines the components that manage the application data in such a way that business-logic processes are separated from data management processes. The WebSA Configuration model defines four stereotypes to define this pattern:
    - *IDataAccessComponent*: Interface of the component that manages the application data. It defines the methods that
    - *DataAccessComponent*: Component that manages the application data.
    - *DataTransferObject*: Data object managed.
    - *BusinessEntityComponent*: Component that manages the business-logic processes.
- **Data Transfer Object** (Fowler, 2002). This pattern establishes a coarse-grain interface between distributed components that facilities the transfer of complete objects as values, thus reducing the number of remote invocations. The WebSA profile includes two stereotypes to define this pattern:
    - *DataTransferObject*: Component that manages the data object.
    - *Assembler*: Component that transform one *DataTransferObject* component into another.
- **Distributed Façade** (Gamma et al., 1995). It establishes a scalable interface between the business logic components and the user interface components, which reduces the coupling degree

between server and client components. The WebSA Configuration model defines two stereotypes for this pattern:

  - *IApplicationFaçade*: Interface that contains the methods offered to the application client.
  - *ApplicationFaçade*: Component that provides access to the business logic processes to the application client.
  - *ApplicationFaçadeProxy*: Component that implements the client to the *ApplicationService* component.

- **Service Gateway** (Trowbridge et al., 2003). This pattern defines a set of components that connect the application to other remote applications, which implement part of the features required. The WebSA Configuration model defines two stereotypes for this pattern:
  - *IServiceGateway*: Interface that defines the methods offered by an external service.
  - *ServiceGateway*: Component that represents a client of an external service.
- **Service Interface** (Trowbridge et al., 2003). It provides an interface to part of the application features that can be used by external clients.
  - *IServiceInterface*: Interface that defines the methods offered to external clients.
  - *ServiceInterface*: Component that provides access to the business logic processes to external clients.

Figure 7.3 illustrates part of the detailed Component model for the SNS SRIA case study (Section 3.4.1, page 54). This example of SRIA architecture is based on the assumption that the SRIA server will be implemented using .NET technologies and, more specifically, the Windows Communication Foundation framework (WCF) and the NHibernate Object-Relational mapping library.

The SRIA server offers three different services to three types of clients, represented by three components with access to the application data and the business-logic processes:

- The *WCFApplicationService* component (*ApplicationFaçade* stereotype, coloured in blue) is the service that provides the access methods for browser or plug-in-oriented RIA user interfaces. This service provides the methods *User_get5tracks*, which retrieves five tracks created by a certain user; *MusicTrack_newTrack*, which creates a new track from the user's data; and *Record_getAllTrackOfRecord*, which retrieves the tracks associated to a certain record. This service uses the *MusicTrackDTO* component as data container, which is created using the *Entity2DTOAssembler* component from the internal *MusicTrackEntity* component. The *MusicTrackDTO* component contains methods for the management of its attributes (*get* and *set*) even though this diagram only shows three of them (*DataTransferObject* components should not modify ID attributes).

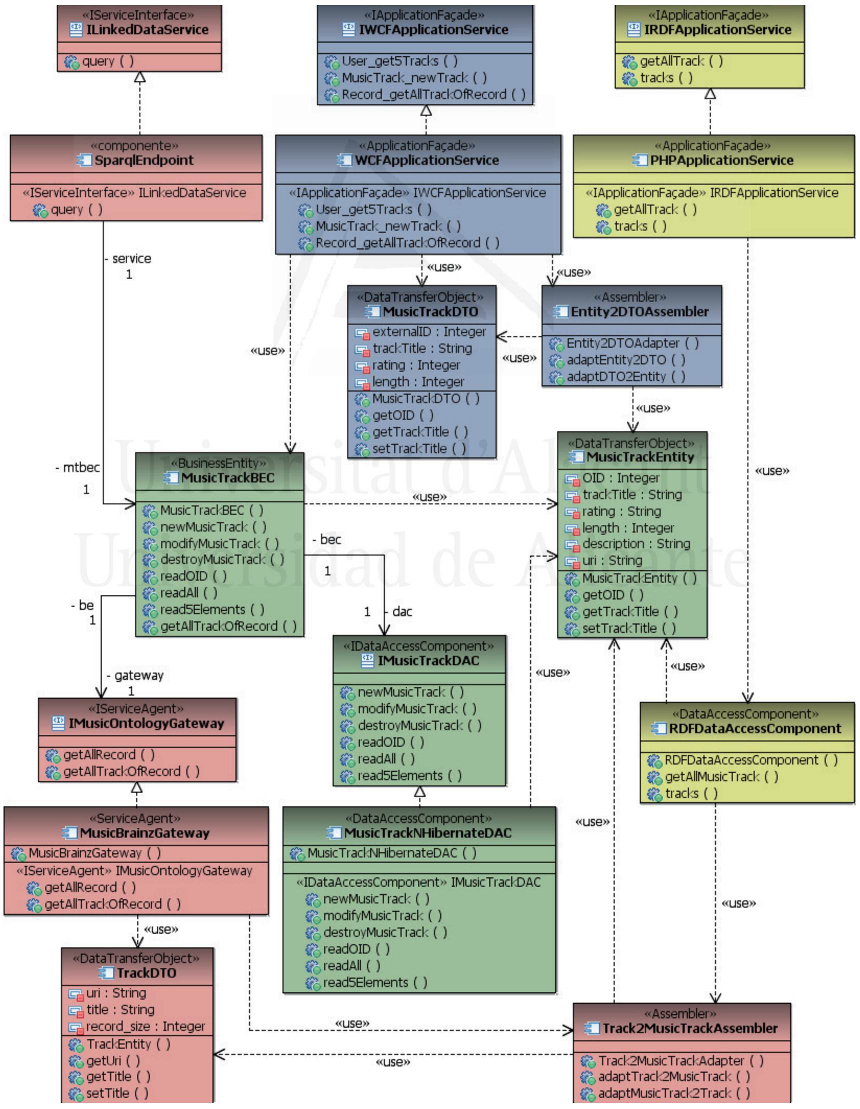


Figure 7.3. Detailed architecture of the SRIA server for the Social Network Site case study.

- The *PHPApplicationService* component (*ApplicationFaçade* stereotype, coloured in yellow) is the service that provides the application data to the *HTML+RDF* view. In this diagram, the service provides two methods: *getAllTrack*, which retrieves all the tracks stored in the database; and *tracks*, which obtains the list of tracks contained in a record.
- The *SparqlEndpoint* component (*ServiceInterface* stereotype, coloured in red) is the service that provides access to the RDF instances of the application to external clients. This service provides a single method called query, which obtains the ontology instances that fulfil the user's query string.

The *WCFApplicationService* and *SparqlEndpoint* components employ several *BusinessEntity* components, which encapsulate the business logic of the application in separate components managing a single data object each. The diagram of the case study only shows the *MusicTrackBEC BusinessEntity* component, which manages the *MusicTrack* objects by means a collection of CRUD methods (*newMusicTrack*, *modifyMusicTrack*, *destroyMusicTrack*, *readOID*, *readAll*, *getAllTrackOfRecord*) and custom methods (*read5Elements*).

BusinessEntity components can employ *DataAccessComponent* components to retrieve or store local data objects from/in the database, and *ServiceAgent* components to retrieve or modify external data objects and invoke external methods. The data objects retrieved are managed as internal *DataTransferObject* components. In the case study, the *MusicTrackBEC* component can use the *MusicTrackNHibernateDAC* component, which manages the *MusicTrack* objects in the database, and the *MusicBrainzGateway* component, which manages the ontology instances stored in the *MusicBrainz* external repository. *ServiceAgent* components work with their own types of *DataTransferObject* components (e.g., *TrackDTO*), which must be converted into internal *DataTransferObject* components before the *BusinessEntity* component use them. This process is performed by an *Assembler* component (e.g., *Track2MusicTrackAssembler*).

The *PHPApplicationService* component employs the *RDFDataAccessComponent* component to access the contents of the local data base and retrieve them as ontology instances.

The implementation of some of the components of the SRIA server could be reused from other applications in order to speed up the development of the server. For instance, at present, some applications, such as the OpenLink Virtuoso server or the D2RQ server, can already offer a SPARQL interface and a HTML view of the data contained in a database from the specification of a set of database-to-ontology mapping rules.

To complete the description of the server architecture, Figure 7.4 and Figure 7.5 show two UML sequence diagrams that describe the process performed after the invocation of two methods of the *WCFApplicationService* component: *User_get5Tracks* and *Record_getAllTrackOfRecord*, respectively. These methods are descriptive examples of the manner in which the SRIA server components work. The first method accesses local data by means of the *MusicTrackNHibernateDAC* component. The process shown in the first diagram could be also performed in traditional RIAs.

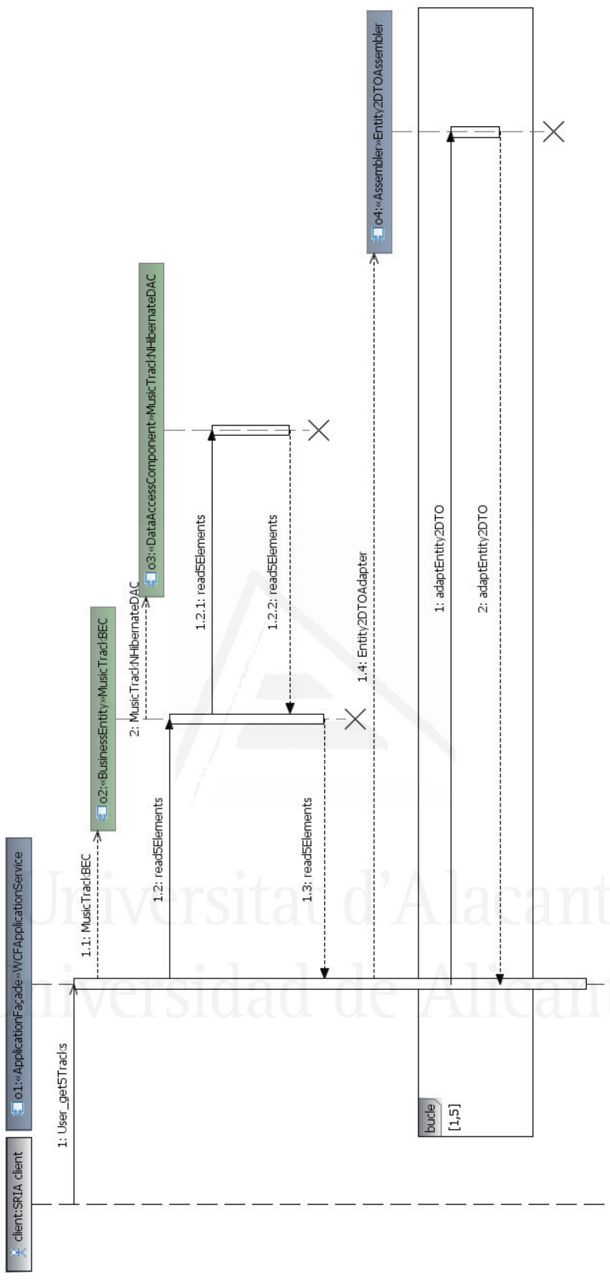


Figure 7.4. UML Sequence diagram of the invocation of the *User_get5Tracks* method of the *WCFApplicationService* component.

The second diagram shows how the service can employ the *MusicBrainzGateway* component in order to retrieve external information about the music tracks stored in the *MusicBrainz* repository.

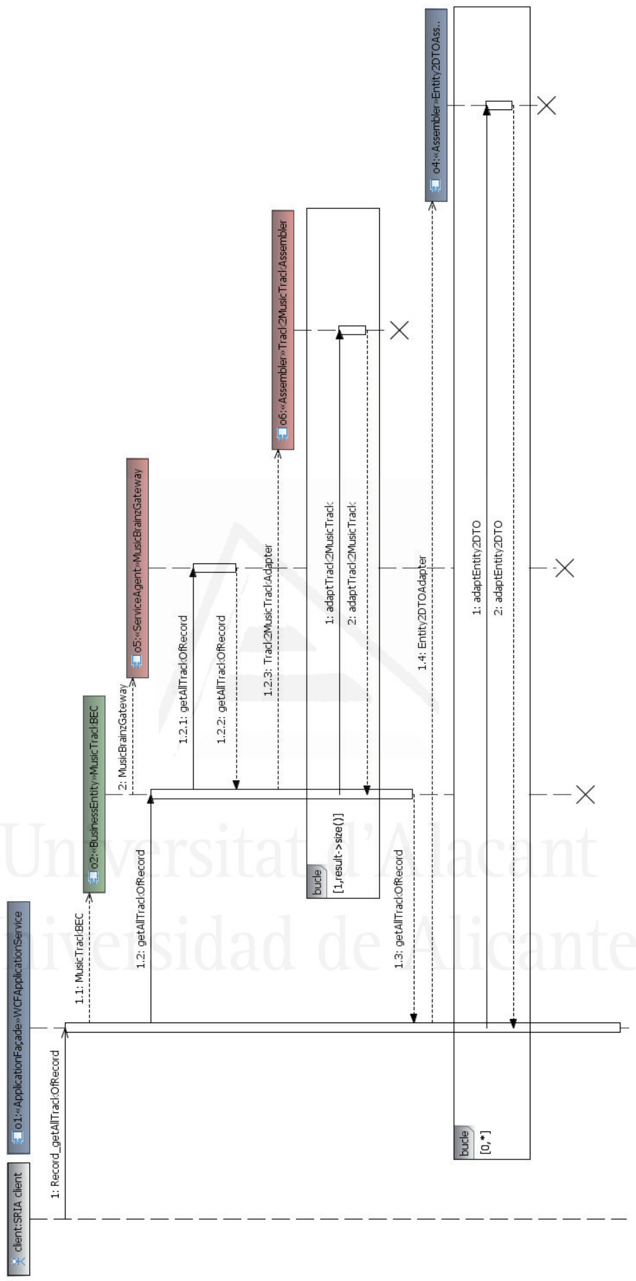

client:SRIA client
o1:«ApplicationFaçade»WCFApplicationService
o2:«BusinessEntity»MusicTrack:BEC
o5:«ServiceAgent»MusicBrainzGateway
o6:«Assembler»Track2MusicTrack:Assembler
o4:«Assembler»Entity2DTOAss...
1: Record_getAllTrackOfRecord
1.1: MusicTrack:BEC
1.2: getAllTrackOfRecord
2: MusicBrainzGateway
1.2.1: getAllTrackOfRecord
1.2.2: getAllTrackOfRecord
1.2.3: Track2MusicTrack:Adapter
1: adaptTrack2MusicTrack
2: adaptTrack2MusicTrack
bucle
[1,result->size()]
1.3: getAllTrackOfRecord
1.4: Entity2DTOAdapter
1: adaptEntity2DTO
2: adaptEntity2DTO
bucle
[0,*]

Figure 7.5. UML Sequence diagram of the invocation of the *Record_getAllTrackOfRecord* method of the *WCFApplicationService* component.

7.1.2 THE ARCHITECTURE OF THE SRIA CLIENT

Figure 7.6 shows the Configuration model of the architecture proposed for the two possible types of SRIA clients. The diagram introduces a general architecture for each of the clients, which could be applied to any case study. The colour scheme chosen corresponds to the one of Figure 7.1 and the component names are coherent with the names of the SRIA modules.

As can be noticed, this diagram contains details about the technology of the client given that the main difference between RIA clients is the technology they are developed on. The diagram represents the components of an AJAX SRIA client (HTML5 and Javascript), as an example of browser-oriented client, and the components of a Silverlight SRIA client, as an example of plug-in-oriented client.

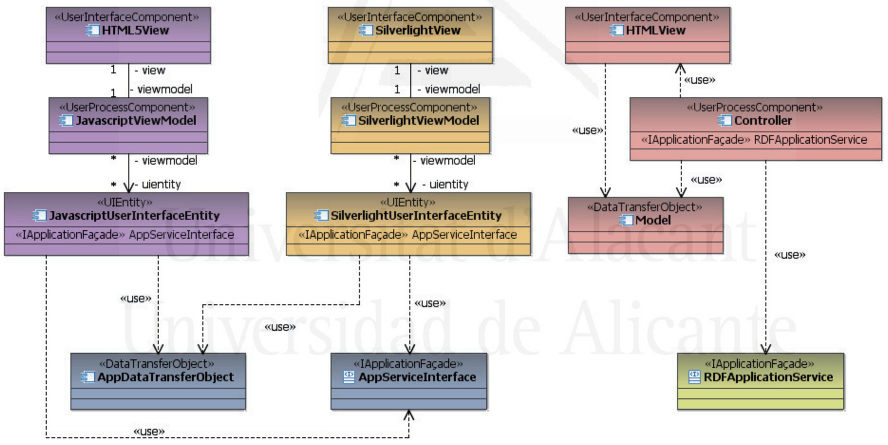

«UserInterfaceComponent» HTML5View
«UserProcessComponent» JavascriptViewModel
«UIEntity» JavascriptUserInterfaceEntity
«IApplicationFaçade» AppServiceInterface
«UserInterfaceComponent» SilverlightView
«UserProcessComponent» SilverlightViewModel
«UIEntity» SilverlightUserInterfaceEntity
«UserInterfaceComponent» HTMLView
«UserProcessComponent» Controller
«IApplicationFaçade» RDFApplicationService
«DataTransferObject» Model
«DataTransferObject» AppDataTransferObject
«IApplicationFaçade» AppServiceInterface
«IApplicationFaçade» RDFApplicationService
- view
- viewmodel
- uientity
«use»

Figure 7.6. Configuration model of the general architecture of two SRIA clients (browser and plug-in oriented).

Table 7.2 shows the components included in all the SRIA server modules, which links the SRIA structure to the components of the Configuration model.

In the same manner than the SRIA server architecture, the following patterns were applied during the design of their architecture in the Configuration model (described by Meliá, 2007):

Table 7.2. Mapping between the SRIA client modules and architectural components.

SRIA client modules		Architectural component
Plug-in-oriented client	*Silverlight client*	SilverlightView, SilverlightViewModel, SilverlightUserInterfaceEntity
	HTML + RDF view	HTMLView, Controller, Model
Browser-oriented client (AJAX)		HTML5View, JavascriptViewModel, JavascriptUserInterfaceEntity

- **Model-View-Controller** (Buschmann et al., 1996). This pattern divides the UI interface in three main components:
    - Model (stereotyped as *DataTransferObject* in the Configuration model. It manages the domain data as the View component requires.
    - View (*UserInterfaceComponent* stereotype). It contains the visualisation elements and depicts the information of the application.
    - Controller (*UserProcessComponent* stereotype). It manages the user input, the communication between client and server components and notifies the View and Model components when they need to change.
- **Model-View-ViewModel** (Gossman, 2005; Smith, 2009). It specialises the Model Presentation pattern (Fowler, 2004) for the .NET platform. However, its terminology and structure has been adopted by other technological solutions, such as iOS or HTML5. In a similar manner than the MVC pattern, it divides the UI in three types of components:
    - Model (stereotyped as *UIEntity* in the Configuration model). It manages the communication with the server components and the local storage of the application data in the UI.

- View (*UserInterfaceComponent* stereotype). It contains the visualisation elements and depicts the information of the application.
- ViewModel (*UserProcessComponent* stereotype). It manages the interaction between users and the View component and the communication between the View and Model components.

The MVVM pattern separates the design from the management of view in two components (view and view model) in such a way that designers can create the view of the application independently from its behaviour and the application data. Moreover, this pattern facilitates the use of processes of lazy synchronisation in the SRIA client for the management of the application data.

The next paragraphs explain in depth two examples of the mentioned architectural patterns for the SNS case study. The first example, representing the MVVM pattern, is illustrated in Figure 7.7, which depicts the WebSA Configuration model of the Silverlight client of the SRIA. In this type of clients, users directly interact with the *UserInterfaceComponent* components (the view), which represent a screenshot of the rich user interface, contain all the widgets of the UI as component attributes, and manage their visualisation (aesthetic features) with the component methods. However, it does not control the information shown. In this example, the *XAMLView* component contains the application widgets, e.g., the *SLStackPanel_MTList* or *SLStackPanel_MTElement* elements of type *StackPanel* (from the Silverlight framework). It also offers a method called *CloneSLStackPanel_MTList* that builds the view of the *SLStackPanel_MTList* (*MusicTrackList*) and populates the element using *SLStackPanel_MTElement* as pattern.

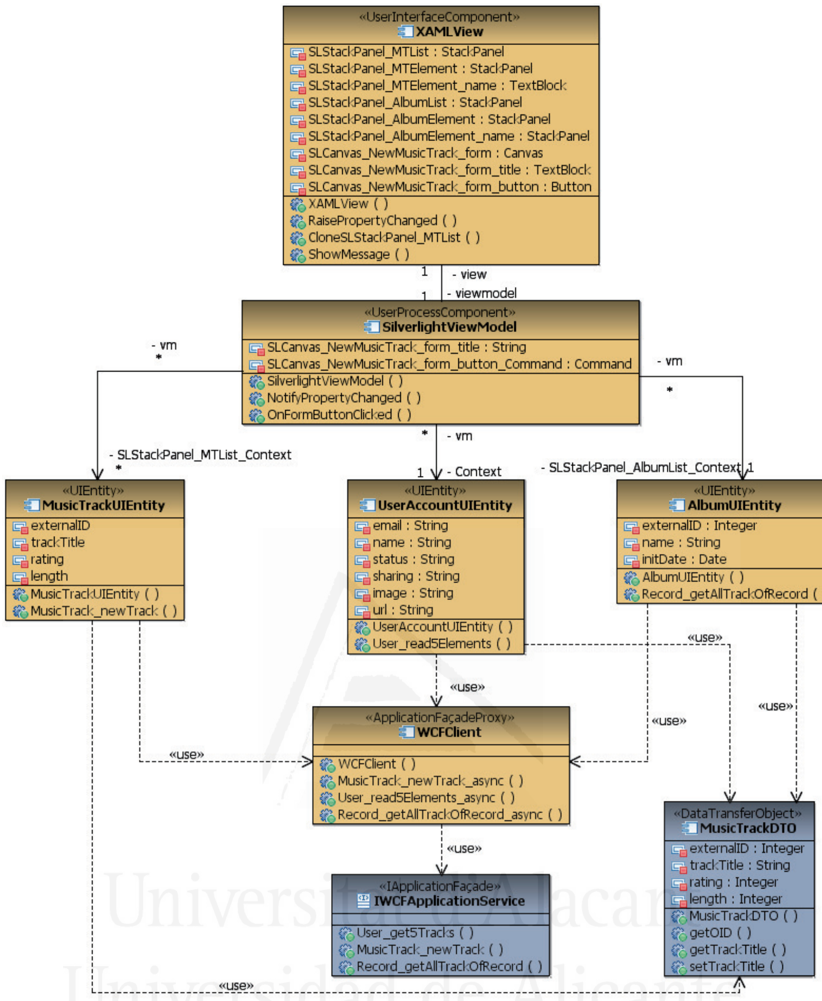

«UserInterfaceComponent» XAMLView
SLStackPanel_MTList : StackPanel
SLStackPanel_MTElement : StackPanel
SLStackPanel_MTElement_name : TextBlock
SLStackPanel_AlbumList : StackPanel
SLStackPanel_AlbumElement : StackPanel
SLStackPanel_AlbumElement_name : StackPanel
SLCanvas_NewMusicTrack_form : Canvas
SLCanvas_NewMusicTrack_form_title : TextBlock
SLCanvas_NewMusicTrack_form_button : Button
XAMLView ( )
RaisePropertyChanged ( )
CloneSLStackPanel_MTList ( )
ShowMessage ( )
- view
- viewmodel
«UserProcessComponent» SilverlightViewModel
SLCanvas_NewMusicTrack_form_title : String
SLCanvas_NewMusicTrack_form_button_Command : Command
SilverlightViewModel ( )
NotifyPropertyChanged ( )
OnFormButtonClicked ( )
- vm
- SLStackPanel_MTList_Context
- Context
- SLStackPanel_AlbumList_Context
«UIEntity» MusicTrackUIEntity
externalID
trackTitle
rating
length
MusicTrackUIEntity ( )
MusicTrack_newTrack ( )
«UIEntity» UserAccountUIEntity
email : String
name : String
status : String
sharing : String
image : String
url : String
UserAccountUIEntity ( )
User_read5Elements ( )
«UIEntity» AlbumUIEntity
externalID : Integer
name : String
initDate : Date
AlbumUIEntity ( )
Record_getAllTrackOfRecord ( )
«use»
«ApplicationFaçadeProxy» WCFClient
WCFClient ( )
MusicTrack_newTrack_async ( )
User_read5Elements_async ( )
Record_getAllTrackOfRecord_async ( )
«DataTransferObject» MusicTrackDTO
externalID : Integer
trackTitle : String
rating : Integer
length : Integer
MusicTrackDTO ( )
getOID ( )
getTrackTitle ( )
setTrackTitle ( )
«IApplicationFaçade» IWCFApplicationService
User_get5Tracks ( )
MusicTrack_newTrack ( )
Record_getAllTrackOfRecord ( )

Figure 7.7. Detailed architecture of the plug-in-oriented SRIA client (Silverlight client) for the SNS case study.

Each *UserInterfaceComponent* component is associated to a single *UserProcessComponent* component (the view-model), which actually manages the information shown by the view, the events triggered by the user and the synchronisation processes between SRIA client and server. Each attribute of the *UserInterfaceComponent* component can be bound to an attribute or a relationship of its corresponding *UserProcessComponent* component. *UserProcessComponent* components manage a collection of UIEntity components, which actually store the information of the data objects used by the view (the model) and the communication processes between SRIA client and server using an *ApplicationFaçadeProxy* component. The proxy component is created from the specification of the interface of the *ApplicationService* component provided by the SRIA server.

In the diagram, the *XAMLView* component is associated to the *SilverlightViewModel* component, which contains two types of attributes:

- Attributes with a simple data type, which are bound to the properties of the attributes of the *XAMLView* component. For instance, the *SLCanvas_NewMusicTrack_form_title* String attribute is bound to the Text property of the *SLCanvas_NewMusicTrack_form_title TextBlock* element.
- Command attributes, which manage the actions performed after an event is triggered. For instance, the *SLCanvas_NewMusicTrack_form_button_Command* attribute is bound to the *Click* event of the *SLCanvas_NewMusicTrack_form_button* attribute.

The *SilverlightViewModel* component employs a collection of *UIEntity* components that manage the communication with the SRIA server and the data objects obtained. Each *UIEntity* component defines a data context, whose data can be visualised by a widget or panel of the *XAMLView* component. In this case, the *SilverlightViewModel* component is associated to the *MusicTrackUIEntity*, *AlbumUIEntity* and *UserAccountUIEntity* components by means of the *SLStackPanel_MTList_Context*, *SLStackPanel_AlbumList_Context* and *Context* relationships. The first two relationships are bound to the *SLStackPanel_MTList* and *SLStackPanel_AlbumList* attributes of the *XAMLView* component, respectively, while the third one is linked to the whole component.

The methods *NotifyPropertyChanged* of the *SilvelightViewModel* component and *RaisePropertyChanged* of the *XAMLView* component are employed to notify the changes in lower layers of the architecture, i.e., the *UIEntity* components and the *SilverlightViewModel* respectively. In this way, the application can perform asynchronous server invocations and maintain the coherence of the data presented to the users.

To complete the description of the architecture of the Silverlight client, Figure 7.8 and Figure 7.9 depict two UML sequence diagrams that represent the behaviour of the client during the invocation of the *Record_getAllTrackOfRecord* and *MusicTrack_newTrack* server methods, respectively. In these diagrams, the SRIA server is considered as another user role.

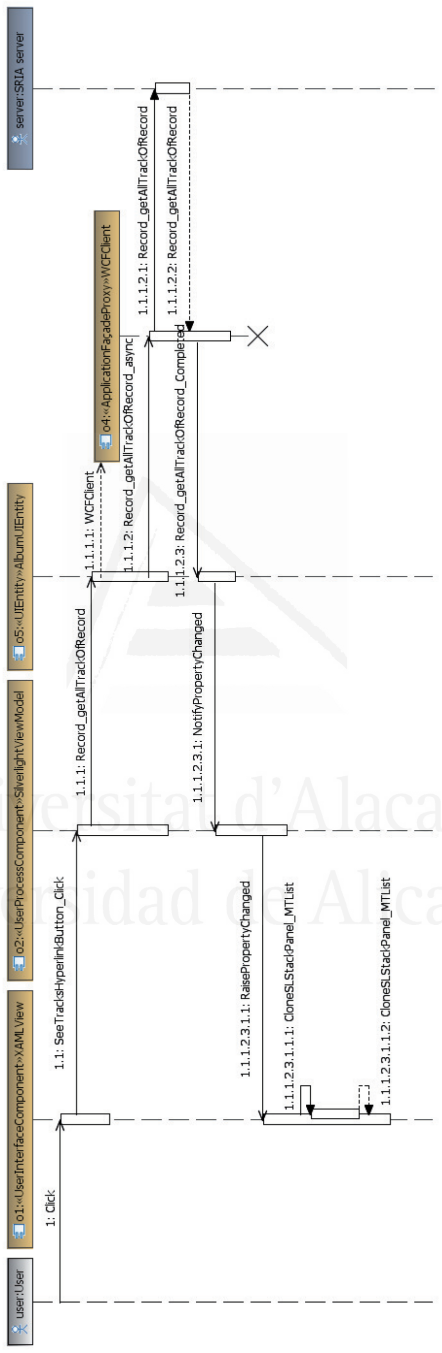

user:User
o1:«UserInterfaceComponent»XAMLView
o2:«UserProcessComponent»SilverlightViewModel
o5:«UIEntity»AlbumUIEntity
o4:«ApplicationFaçadeProxy»WCFClient
server:SRIA server
1: Click
1.1: SeeTracksHyperlinkButton_Click
1.1.1: Record_getAllTrackOfRecord
1.1.1.1: WCFClient
1.1.1.2: Record_getAllTrackOfRecord_async
1.1.1.2.1: Record_getAllTrackOfRecord
1.1.1.2.2: Record_getAllTrackOfRecord
1.1.1.2.3: Record_getAllTrackOfRecord_Completed
1.1.1.2.3.1: NotifyPropertyChanged
1.1.1.2.3.1.1: RaisePropertyChanged
1.1.1.2.3.1.1.1: CloneSLStackPanel_MTList
1.1.1.2.3.1.1.2: CloneSLStackPanel_MTList

Figure 7.8. UML Sequence diagram of the invocation of the *Record_getAllTrackOfRecord* method from the SRIA client.

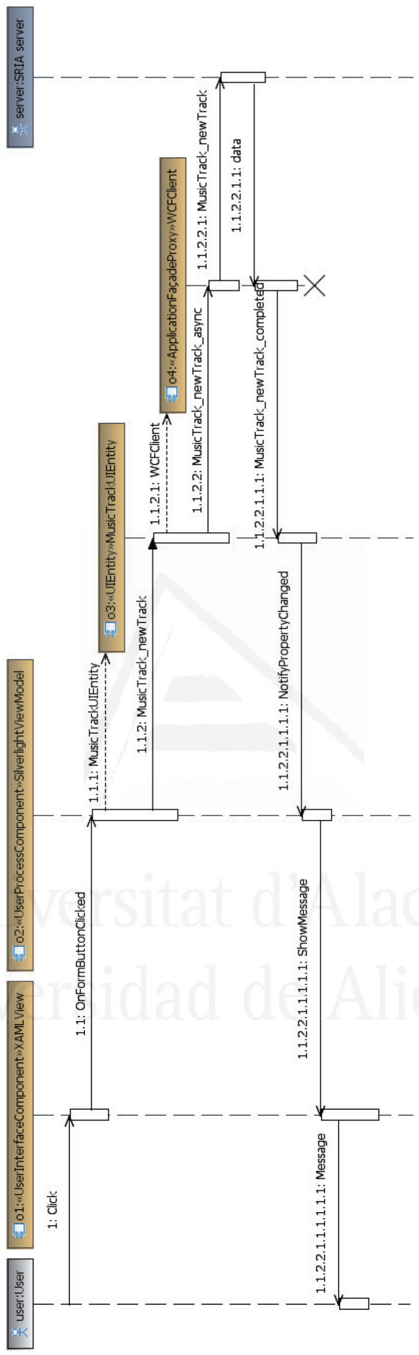

user:User
o1:«UserInterfaceComponent»XAMLView
o2:«UserProcessComponent»SilverlightViewModel
o3:«UIEntity»MusicTrackUIEntity
o4:«ApplicationFaçadeProxy»WCFClient
server:SRIA server
1: Click
1.1: OnFormButtonClicked
1.1.1: MusicTrackUIEntity
1.1.2: MusicTrack_newTrack
1.1.2.1: WCFClient
1.1.2.2: MusicTrack_newTrack_async
1.1.2.2.1: MusicTrack_newTrack
1.1.2.2.1.1: data
1.1.2.2.1.1.1: MusicTrack_newTrack_completed
1.1.2.2.1.1.1.1: NotifyPropertyChanged
1.1.2.2.1.1.1.1.1: ShowMessage
1.1.2.2.1.1.1.1.1.1: Message

Figure 7.9. UML Sequence diagram of the invocation of the MusicTrack_newTrack method from the SRIA client.

The second example of client, which applies the MVC pattern, is illustrated in Figure 7.10. This figure depicts the WebSA Configuration model of the *HTML+RDFView* client of the SRIA. This second client follows the same architectural pattern than most of the traditional user interfaces on the Web. The UI client is composed of several *UserInterfaceComponents* components, i.e., the Web pages, which interact with the users. The *UserProcessComponent* components (the controllers) manage the data visualised by the *UserInterfaceComponent* components and the communication between SRIA server and client. This PHP client uses internal *DataTransferObject* components in order to locally store and manage the data objects obtained from the server

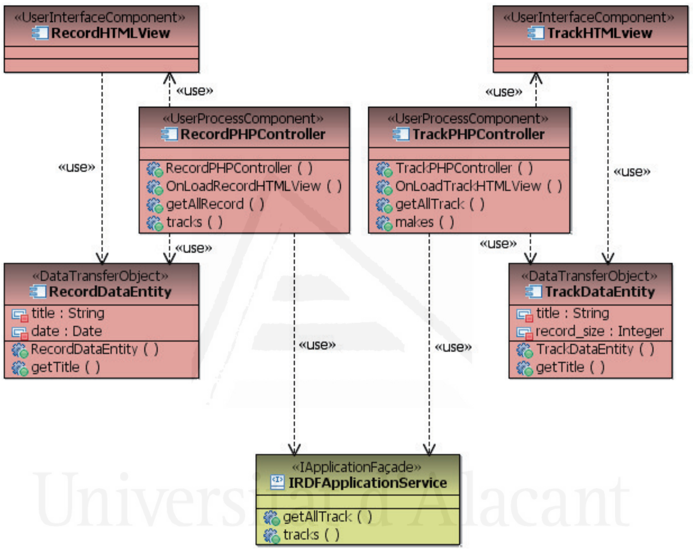


Figure 7.10. Detailed architecture of the plug-in-oriented SRIA client (Silverlight client) for the SNS case study.

As mentioned before, the SRIA architecture is not unique. The one proposed in this section was specified based on the experience gained during the development of the case studies. Using the description of the architecture as a reference, the following subsection describes the transformation rules applied during the third activity of the S^{m}4RIA process.

7.2 MODEL-2-TEXT TRANSFORMATIONS TO OBTAIN A SEMANTIC RIA

In the third activity of the S^{m}4RIA process, model-to-text transformations generate the application code from the information contained in the models. By means of the S^{m}4RIA transformation rules developers can obtain the SRIA software components and resources (mainly ontologies), based on the proposed SRIA architecture. This section addresses the generation of a plug-in-oriented SRIA using technologies from the .NET framework and C# as code language. In particular, the code generated will use the NHibernate framework for the mapping of the database registers into data objects, managed by the application. The Windows Communication Foundation (WCF) framework for the creation of Web services and the Silverlight framework for the development of rich user interfaces.

The schema illustrated in Figure 7.11 specifically shows the SRIA software modules (from Figure 3.2, page 50) and the resources generated from each of the models, which offer an overview of the information captured by each of the models in the previous activities of the S^m4RIA process.

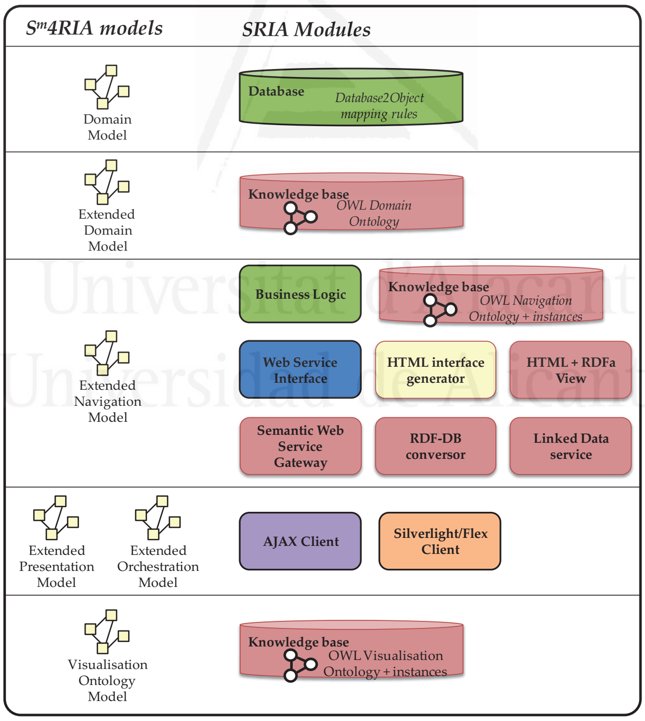


Figure 7.11. SRIA modules and resources obtained from each S^m4RIA model.

The Domain model, the Extended Domain Model and the Extended Navigational Model contain the information for generating all the server-side modules, while the Extended Presentation and Orchestration Models generate the client modules.

- The Domain Model contains the information needed to generate application database, the data objects and the object-relational mapping rules, which turn the data objects into database registers and vice-versa.
- The Extended Domain Model represents the domain ontology, which can be generated in any ontology representation language.
- The Extended Navigational Model is the main server model. As explained before, it represents the information that any type of user can access and the operations that can be invoked by the server client or external agents. From this model, the transformation engines can generate the business logic components, the service interface to the SRIA client and the clients to external Web services. Moreover, they can also obtain the Semantic Web gateway module and the linked data service. The HTML interface generator is also generated from this model since the aesthetic features of this interface are not designed. The HTML interface is oriented to software agents that cannot access the information on the rich user interface (i.e., the Silverlight interface). To this aim, the aspect of the interface is not relevant.

The integration of an existing Database-to-RDF mapping tool, e.g., D2RQ or OpenLink Virtuoso server, can simplify the implementation of the SRIA server architecture and, thus, the definition of the transformation processes and rules. These tools can provide some of the planned functionalities and simplify the resulting applications. For instance, D2RQ (http://www4.wiwiss.fu-berlin.de/bizer/d2rq/) can generate a Linked Data service (based on the SPARQL protocol) and a HTML interface for the visualisation of the ontology instances from the database and a collection of database-2-RDF mapping rules. As illustrated in Figure 7.12, the number of modules to be generated and, in the same manner, the number of transformation rules, can be reduced to a single resource, i.e., a file with the collection of mapping rules mentioned. These rules can be expressed in different languages depending on the tool, e.g., the D2RQ language for the system with the same name. As part of an effort of standardisation of the task, the W3C has recently released the R2RML standard language, which is being adopted by the existing tools.

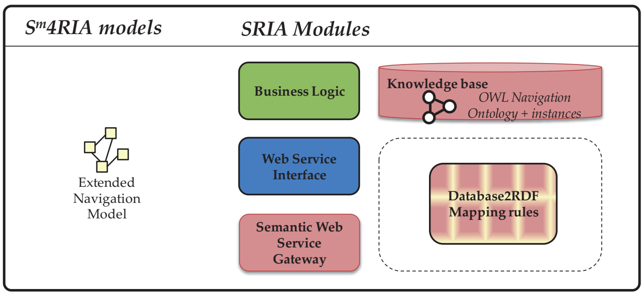

Sm4RIA models
SRIA Modules
Extended Navigation Model
Business Logic
Web Service Interface
Semantic Web Service Gateway
Knowledge base
OWL Navigation Ontology + instances
Database2RDF Mapping rules

Figure 7.12. Modification of the SRIA modules and resources obtained from the Extended Navigational Model.

- The Extended Presentation Model and the Extended Orchestration model can generate a browser-oriented (e.g., Ajax client in the figure) or a plug-in-oriented SRIA client (e.g., Silverlight client in the figure).
- The Visualisation Ontology Model merges the information from the structure and behaviour of the user interface. The Visualisation Ontology and its instances are obtained from this model.

The transformation process defined in the third S^{m}4RIA activity specifies a transformation rule for each of the SRIA modules (e.g., Business Logic) and resources (e.g., ontologies or mapping rules). However, in order to compile the resulting code and deploy the final application, the outcome of the transformation rules must be hierarchically organised in a collection of projects. In this case, the code is organised in a Visual Studio solution, containing several projects that store the architectural components of the application, each in a single file. The solution and the projects also contain a set of configuration files (*sln*, *cproj* or *config* files), external libraries (*dll* files) and auxiliary components, which are also generated during the transformation process and are shared by any application obtained. The chosen file organisation was reused and extended from the OOH4RIA proposal. The description of the structure of the solution generated is introduced in Section G.1 (page 301).

As just mentioned, each transformation rule generates a specific architectural SRIA component in a separate file. Table 7.3 summarises the main model-to-text transformations of the S^{m}4RIA process, the models used as input and the architectural components obtained. The colour scheme used in the second column of the table establishes a relationship between the transformation rules and the SRIA modules and architectural components, depicted in previous figures. The examples of resulting components are obtained from the Configuration models of Figure 7.3 and Figure 7.7. In the same manner, Table 7.4 illustrates a summary of the transformation rules that obtain the main resources of the application.

Table 7.3. Summary of the S^{m}4RIA M2T transformation rules, their input models and the resulting architectural components.

Input Model	Model-to-text transformation rules	Resulting architectural components
Domain *Model*	DEntity_root	`DataTransferComponent, e.g., MusicTrackEntity.`
	Dac_root	`DataAccessComponent,` e.g., MusicTrackDAC.
Extended Domain Model *Extended Navigational Model*	Bec_root	`BusinessEntityComponent,` e.g., MusicTrackBEC.
	OEntity_root	`DataTransferComponent,` e.g., TrackEntity.
	Client_root	`ServiceAgentProxy,` e.g., SparqlClient.
	Gateway_root	`ServiceAgent,` e.g., MusicBrainzGateway.
Extended Navigational Model	Service_root	`ApplicationFaçade,` e.g., WCFApplicationService.
	EEntity_root	`DataTransferComponent,` e.g., MusicTrackDTO.
Extended Presentation Model *Extended Orchestration Model*	View_root	`UserInterfaceComponent,` e.g., XAMLView.
	Viewmodel_root	`UserProcessComponent,` e.g., SilverlightViewModel.
	UIEntity_root	`UIEntity, ApplicationFaçadeProxy,` e.g., MusicTrackUIEntity.
Domain Model *Extended Domain Model* *Extended Navigational Model*	Assembler_root, AssemblerDTOEn_root	`Assembler,` e.g., Entity2DTOAssembler, Track2MusicTrackAssembler.
All	Project_root	Visual Studio project files and auxiliary modules.

Table 7.4. Summary of the S^{m}4RIA M2T transformation rules, their input models and the resulting resources.

Model	Model-to-text transformation rule	Resulting resources
Domain Model	NHibernate_root	NHibernate mapping rules
Extended Domain Model	Ontology_root	Domain ontology
Extended Navigation Model	**Mapping_root**	D2RQ Database-to-RDF mapping rules
	NOntology_root	Navigational ontology Ontology instances
Visualisation Ontology Model	VOntology_root	Visualisation ontology Ontology instances
All	Project_root	Visual Studio project files.

The rules introduced in both tables can be invoked in any order since each of them addresses the creation of different application components. The tables introduce one of the possible organisation of the rules. The rules that employ the Domain Model as input, i.e., *DEntity_root*, *Dac_root* and *NHibernate_root*, are reused from the OOH4RIA methodology. Moreover, the rules for the generation of the SRIA WCF service, i.e., *Service_root* and *EEntity_root*, and the SRIA Silverlight client, i.e., *UIEntity_root*, *Viewmodel_root* and *View_root*, were also imported from OOH4RIA and adapted to employ the new elements of the S^{m}4RIA models. The new transformations included in S^{m}4RIA are depicted in red and yellow, e.g., *OEntity_root*, *Gateway_root*, *DOntology_root* or *Mapping_root*.

All the transformation rules were specified using the Xpand language, which is a template-based language for the definition of model-to-text transformation rules. Unlike QVT, Xpand facilitates the definition of imperative statements, in which designers need to explicitly specify the behaviour of the transformation. With Xpand, designers can define protected regions, in which the code included by users will be safely kept from one generation process to another.

The following subsections will introduce three S^{m}4RIA transformation rules including an example of generated SRIA components: *OEntity_root*, *Gateway_root* and *Mapping_root*. These offer a representative example of the transformations performed in this activity and the manner they are defined using Xpand. The code of other transformation rules can be found in Section F.1 (page 277).

7.2.1 THE *OENTITY_ROOT* MODEL-TO-TEXT TRANSFORMATION

The first example explained is the *OEntity_root* transformation rule, which generates the *DataTransferObject* components (e.g., *TrackEntity*, from Figure 7.3, page 157) used by the *ServiceAgent* components (e.g., *MusicBrainzGateway*) from the information stored in the Extended Navigational Model and the Extended Domain Model. This transformation rule analyses the external navigational classes (*ExtNavigationalClass* metaclass) used in the model and generates one *DataTransferObject* component per each in a separate file using the domain ontology.

The rule also generates the attributes and the operations of the component, specified by the *Concept* class from the EDM which any external navigational class is associated to. When the *Concept* class is also mapped to a domain class, the rule obtains a *DataTransferObject* component that extends the one used by the *DataAccessComponent* component (e.g., *MusicTrackEntity*, from Figure 7.3, page 157). In this case, it adds the attributes required for identifying external objects, i.e., a URI, and the attributes and operations from the *Concept* class with no representation in the Domain model.

Figure 7.13 shows a UML Sequence diagram with the sequence of rule invocation started by the *OEntity_root* rule. This diagram uses a UML profile for defining transformation rules, in which each model-to-text transformation rule is represented as a component and stereotyped as *M2T_Rule*. The invocation of a rule is represented by a synchronous message called *invoke*.

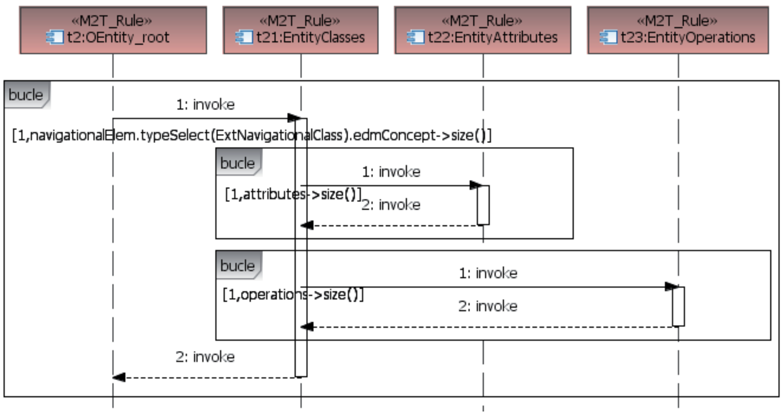

«M2T_Rule» t2:OEntity_root
«M2T_Rule» t21:EntityClasses
«M2T_Rule» t22:EntityAttributes
«M2T_Rule» t23:EntityOperations
bucle
1: invoke
[1,navigationalElem.typeSelect(ExtNavigationalClass).edmConcept->size()]
bucle
1: invoke
[1,attributes->size()]
2: invoke
bucle
1: invoke
[1,operations->size()]
2: invoke
2: invoke

Figure 7.13. UML Sequence diagram of the *OEntity_root* transformation rule.

Table 7.5 contains the Xpand templates for each of the transformation rules shown in the previous figure. The «DEFINE» statements represent the definition of the rules while the «EXPAND» statements their invocations. In order to understand the definition of the rules, Section G.2 (page 302) contains a brief reference of the main elements of the language.

Table 7.5. Xpand code of the *OEntity_root* M2T transformation rule.

```
«DEFINE OEntiy_root FOR EDModel-»
  «EXPAND EntityClasses FOREACH this.models.select(e|e.metaType ==
OntologyModel).elements.typeSelect(Concept)-»
«ENDDEFINE»
```

```
«DEFINE EntityClasses FOR Concept-»
  «LET this.name.toFirstUpper() + "Entity" AS csClassName-»
  «FILE ((String)GLOBALVAR project) + "_LinkedDataCommon" + fileSeparator() +
"Entities" + fileSeparator() + this.model.name.toFirstUpper()+fileSeparator() +
csClassName + ".cs"-»

using System;
namespace «this.getEntityPackage()»
{
    public class «csClassName» «IF this.domainClass != null-»:
«this.domainClass.getENPackage()».«this.domainClass.formattedClassName(getENSuffix()
)»«ENDIF»
    {
        private string __uri = "";
        public string __Uri{ get{ return this.__uri; } set{ this.__uri = value; } }

        «IF this.domainClass == null-»
          «EXPAND EntityAttributes FOREACH this.attributes()-»
          «EXPAND EntityOperations FOREACH this.attributes()-»
        «ENDIF-»
    }
}
  «ENDFILE»
  «ENDLET-»
«ENDDEFINE»
```

```
«DEFINE EntityAttributes FOR Attribute-»
  «LET this.getCsType() AS type-»
  «LET this.name.toFirstLower() AS attrName-»
private «type» «attrName»;
  «ENDLET-»
  «ENDLET-»
«ENDDEFINE»
```

```
«DEFINE EntityOperations FOR Attribute-»
  «LET this.getCsType() AS type-»
  «LET this.name.toFirstLower() AS attrName-»
public «type» «attrName.toFirstUpper()»{ get{ return this.«attrName»; } set{
this.«attrName» = value; } }
  «ENDLET-»
  «ENDLET-»
«ENDDEFINE»
```

The result of this transformation rule can be appreciated in Table 7.6. This table shows two examples of *DataTransferObject* components obtained from the transformation: the *PersonEntity* component, generated from the *Person* concept of the *FOAF* ontology; and the *TrackEntity*, generated from the *Track* concept of the *MusicOntology* ontology (see Figure 5.2, page 88). The *Track* component extends the *MusicTrack DataTransferObject* component employed by the *DataAccessComponent* component by adding the attribute that identifies the external objects, i.e., the URI. In both examples the only operations generated are the setters and getters for each attribute, which, in this case, are implemented as C# properties.

Table 7.6. *PersonEntity* and *TrackEntity DataTransferObject* components, generated by the *OEntity_root* transformation.

```
(FILE: PersonEntity.cs)

using System;

namespace LinkedDataManagement.Entities.FOAF
{
    public class PersonEntity
    {
        private string __uri = "";
        public string __Uri{ get{ return this.__uri; } set{ this.__uri = value; } }

        private string name;
        private string homepage;
        private string email;
        /* Other attributes */
        public string Name{ get{ return this.name; } set{ this.name = value; } }
        public string Homepage{ get{ return this.homepage; } set{ this.homepage =
value; } }
        public string Email{ get{ return this.email; } set{ this.email = value; } }
        /* Other properties & operations*/
    }
}
```

```
(FILE: TrackEntity.cs)

using System;

namespace LinkedDataManagement.Entities.MusicOntology
{
    public class TrackEntity
              : SocialNetworkMDWEGenNHibernate.EN.SocialNetwork.MusicTrackEN
    {
       private string __uri = "";
       public string __Uri { get{ return this.__uri; } set{ this.__uri = value; } }
    }
}
```

7.2.2 THE *GATEWAY_ROOT* MODEL-TO-TEXT TRANSFORMATION

The *Gateway_root* transformation rule generates the *ServiceAgent* components of the application from the Extended Domain Model, which defines the external services and their features, and the Extended Navigation Model, which defines the manner they are used by means of the external navigational classes and external links.

The transformation analyses the use of external navigational classes in the Extended Navigational Model of the application and, according to the domain ontology and the definition of the service, generates a component that can access external information and invoke remote services. Figure 7.14 illustrates a UML sequence diagram with the process of transformation followed (this diagram uses the same UML profile that Figure 7.13). The *Gateway_root* rule invokes a collection of sub-rules that generate the code of the remote invocations and the processing of the results.

The aims of each sub-rule can be briefly described as follows:

- *ServiceGateway*: This rule explores the navigation model searching external navigational classes and links that use a specific external source of the Extended Domain Model and generates the *IServiceAgent* interface and the *ServiceAgent* component for that source (conformant to the interface).
    - *QueryPrefixes*. It creates the prefixes of the SPARQL queries that will be invoked in the remote service. These prefixes will be reused by any SPARQL request.
    - *LinkMap*. It creates the operations that invoke the external services from the external traversal or service links.

- *LinkArguments*. It generates the arguments of the operation generated by the *LinkMap* rule.
- *QueryMap*. It generates a SPARQL sentence for each traversal or service link.
- *ResultMap*. It generates the code that maps the data objects obtained from the external service into a *DataTransferObject* component that could be managed by the application (i.e., those generated by the *OEntity_root* transformation).

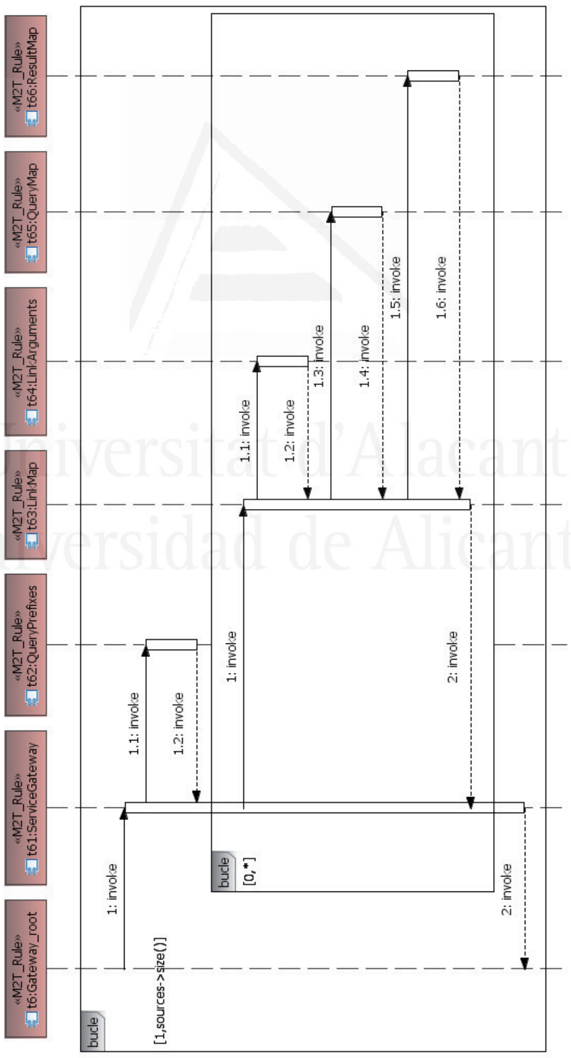


Figure 7.14. UML Sequence diagram of the Gateway_root M2T transformation rule.

Table 7.7 contains the Xpand templates for each of the transformation rules shown in the previous figure. The «PROTECT» statement (see *QueryMap* rule) defines a protected region, in which the code modified by the developers will be kept safely between generation processes.

Table 7.7. Xpand code of the *Gateway_root* M2T transformation rule.

```
«DEFINE Gateway_root FOR ENModel-»
  «FOREACH ((List[ExtNavigationalClass]) navigationalElem.select(e | e.metaType
== ExtNavigationalClass)).source.toSet() AS source-»
    «EXPAND ServiceGateway-»
  «ENDFOREACH-»
«ENDDEFINE»
```

```
«DEFINE ServiceGateway (Source source) FOR ENModel-»
    «LET navigationalElem.typeSelect(TravesalLink).select(l |
l.nodeTarget.metaType == ExtNavigationalClass && ((ExtNavigationalClass)
l.nodeTarget).source == source) AS externalLinks-»
    «LET ((String)GLOBALVAR project) + "_LinkedDataCommon" + fileSeparator() +
"Gateways" + fileSeparator() + source.edModel.name.toFirstUpper() +
fileSeparator() AS path-»

«REM» *** SERVICE AGENT INTERFACE *** «ENDREM»
    «FILE path + "I" + source.name.toFirstUpper() + "ServiceGateway.cs"-»
using System;
using System.Net;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using Newtonsoft.Json.Linq;
    «REM» Add DTO using statements «ENDREM»
namespace LinkedDataManagement.«source.edModel.name.toFirstUpper()».Gateways
{
    public partial interface I«source.name.toFirstUpper()»ServiceGateway
    {
         «EXPAND methodSignature FOREACH externalLinks-»
    }
}
    «ENDFILE-»

«REM» *** SERVICE AGENT COMPONENT *** «ENDREM»
    «FILE path + source.name.toFirstUpper() + "ServiceGateway.cs"-»
using System;
using System.Net;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using Newtonsoft.Json.Linq;
    «REM» Add DTO using statements «ENDREM»
using LinkedDataManagement.«source.edModel.name.toFirstUpper()».Clients;
```

```
namespace LinkedDataManagement.«source.edModel.name.toFirstUpper()».Gateways
{
    public partial class «source.name.toFirstUpper()»ServiceGateway :
I«source.name.toFirstUpper()»ServiceGateway
    {
        private const string PREFIXES = @"PREFIX owl:
<http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
«EXPAND QueryPrefixes FOREACH this.edModel.models-»";

        public «source.name.toFirstUpper()»ServiceGateway()
        {
        }

«EXPAND LinkMap FOREACH externalLinks-»
    }
}
    «ENDFILE-»
    «ENDLET-»
    «ENDLET-»
«ENDDEFINE»
```

```
«DEFINE QueryPrefixes FOR OntologyModel-»
   PREFIX «namespace»: <«uriBase»>
«ENDDEFINE»
```

```
«DEFINE LinkMap FOR TravesalLink-»
  «LET (ExtNavigationalClass)this.nodeTarget AS externalClass-»
  «LET externalClass.edmConcept.name.toFirstUpper() + "Entity" AS enClass-»
        public IList<«enClass»> «name.toFirstUpper()»_«((NavigationalModel)
GLOBALVAR enModel).formattedName()»(«EXPAND LinkArguments»)
        {
            List<«enClass»> list = null;
            string query = PREFIXES + @"«EXPAND QueryMap»";

            // Invoke the remote service
            «externalClass.source.name»SparqlClient sparqlClient = new
«externalClass.source.name-»SparqlClient();

            list = new List<«enClass»>();
            try
            {
                string result = sparqlClient.Query( query );
                // Parse the resulting JSON object
                JObject json = JObject.Parse(result);
                JArray resultArray = (JArray) json["results"]["bindings"];
                // Process the results
                foreach (JToken token in resultArray.Children())
                {
                    var track = new «enClass»();
                    track.__Uri = (string)token["uri"]["value"];
                    «IF externalClass.edmConcept.domainClass != null &&
externalClass.edmConcept.domainClass.dataTypeOID() == PrimitiveType::String-»
track.«externalClass.edmConcept.domainClass.getOIDProperty().toFirstUpper()-» =
(string)token["uri"]["value"];
                    «ENDIF-»

                    «EXPAND ResultMap FOREACH externalClass.navAttribute.
typeSelect(ExternalNavigationalAttribute)-»
                    list.Add(track);
                }
            }
            catch
            {
                list = null;
            }
            return list;
        }
  «ENDLET-»
  «ENDLET-»
«ENDDEFINE»
```

```
«DEFINE LinkArguments FOR TravesalLink-»
  «LET ((List[String]) List[String].newInstance()) AS argList-»
  «IF paging»«IF argList.add("int offset").add("int limit")!=null»«ENDIF»«ENDIF»
  «IF this.metaType == ExternalTraversalLink && ((ExternalTraversalLink)this).
edmObjectProperty != null»
    «IF argList.add("String uriParam") != null»«ENDIF»
  «ENDIF-»
  «FOREACH argList AS elem SEPARATOR ","»«elem»«ENDFOREACH-»
  «ENDLET-»
«ENDDEFINE»
```

```
«DEFINE QueryMap FOR TravesalLink-»
  «LET ((ExtNavigationalClass) nodeTarget) AS target-»
SELECT ?uri «FOREACH
target.navAttribute.typeSelect(ExternalNavigationalAttribute) AS a-»?«a.name»
«ENDFOREACH»
WHERE
{
   ?uri rdf:type «target.edmConcept.model.namespace»:«target.edmConcept.name» .
   «REM» Expand conditions for property navigation «ENDREM»
   «IF this.metaType == ExternalTraversalLink && ((ExtNavigationalClass)
this.nodeOrigin).source == ((ExtNavigationalClass) this.nodeTarget).source-»
      «LET ((ExternalTraversalLink)this).edmObjectProperty AS objProperty-»
      «IF objProperty != null-»
      <" + uriParam + @"> «objProperty.getNamespace()»:«objProperty.name» ?uri .
      «ENDIF-»
      «ENDLET-»
    «ENDIF-»

    «REM»Expand conditions related to attributes«ENDREM»
    «FOREACH target.navAttribute.typeSelect(ExternalNavigationalAttribute) AS a»
      «IF a.edmAttribute.metaType == RefAttribute-»
      «LET (RefAttribute) a.edmAttribute AS refA-»
   OPTIONAL { ?uri «refA.refProperty.getNamespace()»:«refA.refProperty.name»
?«a.name» } .
```

```
      «ENDLET-»
      «ELSE-»
   OPTIONAL { ?uri «a.edmAttribute.getNamespace()»:«a.edmAttribute.name»
?«a.name» } .
      «ENDIF-»
    «ENDFOREACH-»
}
  «IF paging-»
OFFSET " + offset + " LIMIT " + limit + @"
  «ELSE-»
LIMIT 5
  «ENDIF-»
«ENDLET-»
«ENDDEFINE»
```

```
«DEFINE QueryMap FOR ExternalServiceLink-»
  SELECT ?uri «FOREACH ((NavigationalClass)
nodeTarget).navAttribute.typeSelect(ExternalNavigationalAttribute) AS a-
»?«a.name» «ENDFOREACH»
  WHERE
  {
     ?uri rdf:type «((ExtNavigationalClass)
nodeTarget).edmConcept.model.namespace»:«((ExtNavigationalClass)
nodeTarget).edmConcept.name» .
     «FOREACH ((NavigationalClass)
nodeTarget).navAttribute.typeSelect(ExternalNavigationalAttribute) AS attr-»
     ?uri «attr.edmAttribute.getNamespace()»:«attr.edmAttribute.name»
?«attr.name» .
     «ENDFOREACH»

     «REM»Generate SPARQL filters for each argument«ENDREM»
     «PROTECT CSTART "#*" CEND "*#" ID getFilterRegionId() -»
     «FOREACH argumentLink.typeSelect(ExternalArgumentLink).select(e |
extendedDomainModel::Attribute.isInstance(e.edmElement)) AS arg-»
        «LET (extendedDomainModel::Attribute) arg.edmElement AS attr-»
        ?var «attr.getNamespace()»:«attr.name» ?«attr.name» .
        «IF ((extendedDomainModel::Attribute) arg.edmElement).target ==
XmlDatatypes::string-»
        FILTER regex(?«attr.name», "^«arg.value»").
        «ELSEIF ((extendedDomainModel::Attribute) arg.edmElement).target ==
XmlDatatypes::integer-»
        FILTER (?«attr.name» = «arg.value»)
        «ENDIF»
        «ENDLET»
     «ENDFOREACH»
     «ENDPROTECT»
  }
«ENDDEFINE»
```

Table 7.8 contains the result of this transformation for the MusicBrainz source, which contains instances of the MusicOntology ontology (see Figure 5.2, page 88). The table shows the code of the *MusicBrainzServiceGateway* component with the

GetAllRecord_SocialNetwork operation, which retrieves a list of albums from the MusicBrainz repository. The operation employs the RecordEntity component, generated by the *OEntity_root* rule, and the *MusicBrainzSparqlClient* component, generated by the *Client_root* rule.

Table 7.8. *ServiceAgent* component for the MusicBrainz service generated with the *Gateway_root* transformation rule.

```
using System;
using System.Net;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using Newtonsoft.Json.Linq;
using LinkedDataManagement.Entities.MusicOntology;
using LinkedDataManagement.Default.Clients;

namespace LinkedDataManagement.Default.Gateways
{
    public partial class MusicBrainzServiceGateway : IMusicBrainzServiceGateway
    {
        private const string PREFIXES = @"PREFIX owl:
<http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX socialnetwork: <http://socialnetwork.com>
PREFIX : <>
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
";
        public MusicBrainzServiceGateway()
        {
        }

        public IList<RecordEntity> GetAllRecord_SocialNetwork()
        {
            List<RecordEntity> list = null;

            string query = PREFIXES + @"  SELECT ?uri ?title
WHERE
{
        ?uri rdf:type mo:Record .
        OPTIONAL { ?uri dc:title ?title } .
}
LIMIT 20
";
            // Invoke the remote service
            MusicBrainzSparqlClient sparqlClient = new MusicBrainzSparqlClient();

            list = new List<RecordEntity>();

            try
            {
                string result = sparqlClient.Query( query );

                // Parse the resulting JSON object
                JObject json = JObject.Parse(result);
                JArray resultArray = (JArray) json["results"]["bindings"];

                // Process the results
                foreach (JToken token in resultArray.Children())
                {
                    var track = new RecordEntity();

                    track.__Uri = (string)token["uri"]["value"];
                    try
                    {
                        track.Name = (string) token["title"]["value"];
                    }
                    catch { }

                    list.Add(track);
                }
            }
            catch
            {
                list = null;
            }
            return list;
        }
    }
}
```

7.2.3 THE *MAPPING_ROOT* MODEL-TO-TEXT TRANSFORMATION

The last example of this section is the Mapping_root transformation rule, which generates the database-to-RDF mapping rules for the D2RQ system using their own language. From the information contained in the Semantic Web Agent view of the Extended Navigational Model (see Figure 5.9, page 111) and the Extended Domain Model, this rule generates the mapping rules for this system.

The rule analyses the external navigational classes and traversal links contained in the view and generates the required mapping rules. It also generates the statements for the setup of the database and the Web server in a protected block, which can be safely modified by the developers. Figure 7.15 illustrates a UML sequence diagram that describes the transformation process followed in this rule (it is conformant to the same UML profile that the previous diagrams).

The tasks performed by each sub-rule in the diagram can be described as follows:

- *Mapping_root*: This rule manages the generation of the mapping rules and generates the setup data for the database connection and the ServiceInterface architectural components provided, i.e., the *SparqlEndpoint* and the *PHPServiceInterface* components (from Figure 7.3, page 157).

- *OntologyModelPrefix*: This rule generates the URI prefixes required for the mapping file from the models described in the Extended Domain Model.
- *ConceptMap*: This rule creates the D2RQ mapping rule for the concepts associated to the external navigational classes of the Extended Navigational Model.
- *AssociationMap*: This rule creates the D2RQ mapping rule for the associations linked to the external traversal links of the Extended Navigational Model.
- *PropertyMap*: This rule generates the D2RQ mapping rule for the associations linked to the external traversal links of the Extended Navigational Model.

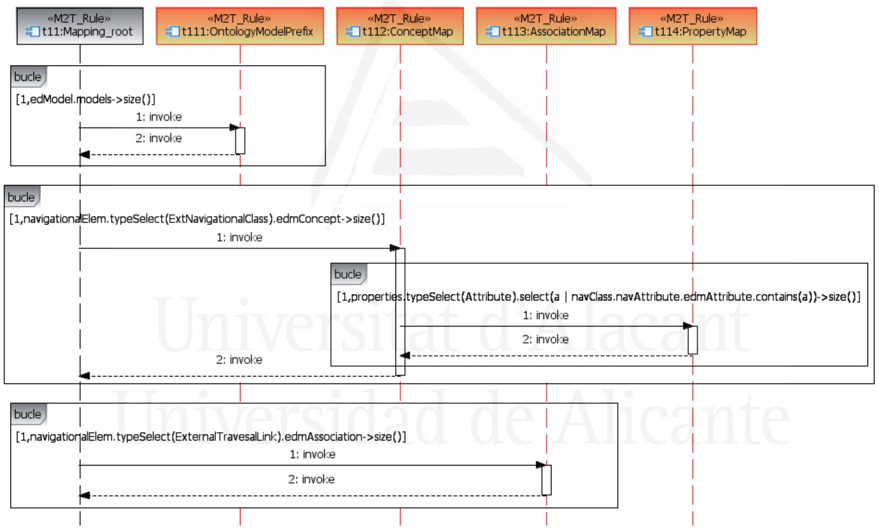

«M2T_Rule» t11:Mapping_root
«M2T_Rule» t111:OntologyModelPrefix
«M2T_Rule» t112:ConceptMap
«M2T_Rule» t113:AssociationMap
«M2T_Rule» t114:PropertyMap
bucle
[1,edModel.models->size()]
1: invoke
2: invoke
bucle
[1,navigationalElem.typeSelect(ExtNavigationalClass).edmConcept->size()]
1: invoke
bucle
[1,properties.typeSelect(Attribute).select(a | navClass.navAttribute.edmAttribute.contains(a))->size()]
1: invoke
2: invoke
2: invoke
bucle
[1,navigationalElem.typeSelect(ExternalTravesalLink).edmAssociation->size()]
1: invoke
2: invoke

Figure 7.15. UML sequence diagram of the *Mapping_root* transformation rule.

Table 7.9 shows the Xpand code for each of the sub-rules. The complete syntax of the D2RQ mapping rules can be found at the D2RQ Web site[45]. The D2RQ language defines each statement as a RDF triple (subject, predicate, object) using the N3 format. Mapping rules are collections of these triples. When a triple is ends in ";", the subsequent triple shares the same subject. In Xpand, the «`GLOBALVAR`» statement is used for defining global variables, external to the models and more related to the generation engine. In this case, some information about the

[45] The D2RQ mapping language: `http://d2rq.org/d2rq-language`

database has been provided externally in order to facilitate the reuse of the same model in different projects.

Table 7.9. Xpand code of the *Mapping_root* model-to-text transformation rule.

```
«DEFINE Mapping_root FOR ENModel-»
   «FILE name.toLowerCase() + ".n3"-»
@prefix map: <file:/C:/Users/«name.toLowerCase()».n3#> .
@prefix db: <> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .
@prefix d2r: <http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2r-server/config.rdf#> .

«EXPAND OntologyModelPrefix FOREACH edModel.models»

«PROTECT CSTART "#*" CEND "*#" ID "ServerConfig"»
# SERVER CONFIGURATION
#
<> a d2r:Server;
    rdfs:label "HTML + RDFa view";
    d2r:baseURI <http://localhost:2020/>;
    d2r:port 2020;
    d2r:vocabularyIncludeInstances true;
#    d2r:metadataTemplate "metadata.n3";
#    d2r:documentMetadata [
#        rdfs:comment "This comment is custom document metadata.";
#    ];
    .

map:Configuration a d2rq:Configuration;
        d2rq:useAllOptimizations true
        .
«ENDPROTECT»

# DATABASE CONFIGURATION
#
map:database a d2rq:Database;
    # SQL Server
        d2rq:jdbcDriver "com.microsoft.sqlserver.jdbc.SQLServerDriver";
        d2rq:jdbcDSN
"jdbc:sqlserver://localhost:1405;instanceName=sqlexpress;databaseName=«GLOBALVAR
projectName»NHibernate";
        d2rq:username "nhibernateUser";
        d2rq:password "nhibernatePass";
        .


    «EXPAND OntologyConceptMap FOREACH
navigationalElem.typeSelect(ExtNavigationalClass).edmConcept»
    «EXPAND OntologyAssociationMap FOREACH
```

```
navigationalElem.typeSelect(ExternalTraversalLink).edmAssociation»


   «ENDFILE»
«ENDDEFINE»
```

```
«DEFINE OntologyModelPrefix FOR OntologyModel-»
@prefix «namespace»: <«uriBase»«IF isLocal && !uriBase.endsWith("/")»#«ENDIF»> .
«ENDDEFINE»
```

```
«DEFINE ConceptMap(ExtNavigationalClass navClass) FOR Concept-»
     map:«name» a d2rq:ClassMap;
          d2rq:dataStorage map:database;
          d2rq:uriPattern "«name»/@@«GLOBALVAR db».«domainClass.alias».
«this.domainClass.firstIdentifier().alias»|urlify@@";
          d2rq:class «model.namespace»:«name»;
          d2rq:classDefinitionLabel "«domainClass.name»";
          .
«EXPAND PropertyMap FOREACH properties.select(p|p.visibility==Visibility::Public &&
navClass.navAttribute.typeSelect(ExternalNavigationalAttribute).edmAttribute.contai
ns(p))»
«ENDDEFINE»
```

```
«DEFINE PropertyMap FOR Property-»«ENDDEFINE»


«DEFINE PropertyMap FOR Attribute-»
map:«concept.name»_«name» a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:«concept.name»;
        d2rq:property «concept.model.namespace»:«name»;
        d2rq:propertyDefinitionLabel "«concept.name» «name»";
        d2rq:column "«GLOBALVAR
db».«concept.domainClass.alias».«domainAttribute.alias»";
        .
    «ENDDEFINE»
```

```
«DEFINE AssociationMap FOR Association-»
map:«conceptOrigin.name»_«direct.name» a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:«conceptOrigin.name»;
        d2rq:property «conceptOrigin.model.namespace»:«direct.name»;
        d2rq:refersToClassMap map:«conceptTarget.name»;
        «PROTECT CSTART "#*" CEND "*#" ID getDirectId(this.direct)»
        # TABLE JOINS
        «ENDPROTECT»
        .
        «IF inverse != null»
map:«conceptTarget.name»_«inverse.name» a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:«conceptTarget.name»;
        d2rq:property «conceptTarget.model.namespace»:«conceptTarget.name»;
        d2rq:refersToClassMap map:«conceptOrigin.name»;
        «PROTECT CSTART "#*" CEND "*#" ID getInverseId(this.inverse)»
        # TABLE JOINS
        «ENDPROTECT»
        .
        «ENDIF»
«ENDDEFINE»
```

```
«DEFINE AssociationMap FOR LinkedAssociation-»
  «LET conceptOrigin.domainClass.alias AS originTableName-»
  «LET (conceptOrigin.domainClass.alias == conceptTarget.domainClass.alias &&
!direct.domainAttribute.isManyToMany() ? conceptTarget.domainClass.alias + "Alias"
: conceptTarget.domainClass.alias )  AS targetTableName-»
map:«conceptOrigin.name»_«direct.name» a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:«conceptOrigin.name»;
    d2rq:property «direct.concept.model.namespace»:«direct.name»;
    d2rq:refersToClassMap map:«conceptTarget.name»;
    #DIRECT
    «IF direct.domainAttribute.isOneToMany()-»
    d2rq:join "«GLOBALVAR db».«originTableName».«conceptOrigin.domainClass.
firstIdentifier().alias» <= «GLOBALVAR
db».«targetTableName».«direct.domainAttribute.associationOtherSide().getFKName()»";
    «ELSEIF direct.domainAttribute.isManyToOne()-»
    d2rq:join "«GLOBALVAR db».«originTableName».«direct.domainAttribute.
getFKName()» => «GLOBALVAR db».«targetTableName».«conceptTarget.domainClass.
firstIdentifier().alias»";
    «ELSEIF direct.domainAttribute.isManyToMany()-»
    d2rq:join "«GLOBALVAR db».«originTableName».«conceptOrigin.domainClass.
firstIdentifier().alias» <= «GLOBALVAR db».«direct.domainAttribute.association()
.alias».«direct.domainAttribute.associationOtherSide().getFKName()»";
    d2rq:join "«GLOBALVAR db».«direct.domainAttribute.association().alias»
.«direct.domainAttribute.getFKName()» => «GLOBALVAR
db».«targetTableName».«conceptTarget.domainClass.firstIdentifier().alias»";
    «ELSEIF direct.domainAttribute.isOneToOne()-»
    d2rq:join "«GLOBALVAR db».«originTableName».«direct.domainAttribute.
getFKName()» => «GLOBALVAR db».«targetTableName».«conceptTarget.domainClass.
firstIdentifier().alias»";
    «ELSE-»
        # ERROR
    «ENDIF-»
    «IF conceptOrigin.domainClass.alias == conceptTarget.domainClass.alias &&
!direct.domainAttribute.isManyToMany()-»
        d2rq:alias "«originTableName» AS «targetTableName»";
    «ENDIF-»
        .

    «IF inverse != null-»
map:«conceptTarget.name»_«inverse.name» a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:«conceptTarget.name»;
    d2rq:property «inverse.concept.model.namespace»:«inverse.name»;
    d2rq:refersToClassMap map:«conceptOrigin.name»;
    #INVERSE
                «REM»d2rq:join sentences«ENDREM»
    «ENDIF-»
  «ENDLET-»
  «ENDLET-»
«ENDDEFINE»
```

Table 7.10 illustrates part of the code resulting from the transformation rule for the Social Network case study. In this case, the table shows the configuration parameters of the server interfaces (triple

"`<> a d2r:Server;`", within a protected block) and the database (triple "`map:database a d2rq:Database;`") and the mapping rules for generating the information of the users as instances of the FOAF ontology (the name of the users will be generated as instances of the *foaf:name* property).

Table 7.10. Database-to-RDF mapping rules for the Social Network case study using the D2RQ language.

```
@prefix map: <file:/C:/Users/a.n3#> .
@prefix db: <> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
# MORE PREFIXES
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .
@prefix d2r: <http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2r-server/config.rdf#> .
@prefix foaf: <http://xlmns.com/foaf/0.1/> .
@prefix sns: <http://www.dlsi.ua.es/sns#> .

#*PROTECTED REGION ID(ServerConfig) ENABLED START*#
# SERVER CONFIGURATION
#
<> a d2r:Server;
    rdfs:label "HTML + RDFa view";
    d2r:baseURI <http://localhost:2020/>;
    d2r:port 2020;
    d2r:vocabularyIncludeInstances true;
#    d2r:metadataTemplate "metadata.n3";
#    d2r:documentMetadata [
#        rdfs:comment "This comment is custom document metadata.";
#    ];
    .

map:Configuration a d2rq:Configuration;
        d2rq:useAllOptimizations true
        .
#*PROTECTED REGION END*#

# DATABASE CONFIGURATION
#
map:database a d2rq:Database;
# SQL Server
        d2rq:jdbcDriver "com.microsoft.sqlserver.jdbc.SQLServerDriver";
        d2rq:jdbcDSN
"jdbc:sqlserver://localhost:1405;instanceName=sqlexpress;databaseName=NHibernate";
        d2rq:username "nhibernateUser";
        d2rq:password "nhibernatePass";
        .

map:User a d2rq:ClassMap;
         d2rq:dataStorage map:database;
         d2rq:uriPattern "User/@@SocialNetwork.UserAccount.email|urlify@@";
         d2rq:class foaf:Person;
         d2rq:classDefinitionLabel "UserAccount";
         .

map:User_name a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:User;
        d2rq:property foaf:name;
        d2rq:propertyDefinitionLabel "User name";
        d2rq:column "SocialNetwork.UserAccount.name";
        .
# OTHER MAPPING RULES
```

These examples have provided an overview of the transformation processes performed in the third activity of the S^{m}4RIA process. As mentioned before, the code of other model-to-text transformation rules can be found in Section F.1 (page 277). After this activity, the developers of the SRIA obtain a Visual Studio project with all the code generated and the OWL ontologies and mapping rules. They need to compile the code and deploy the server and client components into a Microsoft Internet Information Services server[46].

Those components provided by D2RQ should be downloaded from the D2RQ Web site[47], unzipped and initialised as the D2RQ developers indicate using the mapping file generated. These components can be deployed to an Apache Tomcat server[48] as a Java Web service or can be directly executed by the D2RQ framework.

The subsequent subsection explains the model-to-model transformations designed to speed up the development of a SRIA with the S^{m}4RIA methodology. They automatically generate mock-ups of the main models, which can be extended and/or modified by the designers.

7.3 Model-to-Model Transformations to Accelerate the S^{m}4RIA Process

Model-to-model transformation rules transform the elements of one or more origin models (CIM, PIM or PSM) into the elements of one or more target models. This type of transformations facilitates the creation of mock-ups of models, which should be modified or completed by the designers, or even complete models. This last approach can accelerate the design processes of the methodology and thus reduce the time and effort spent in the development.

As explained in Section 4.2.3 (page 69), the modelling tasks in the S^{m}4RIA process are mostly performed by human designers, which usually need to create all the elements of a model and linked them to others. Some of these tasks can be automated partially or totally by means of model-to-model transformations. This section describes the

[46] The Official Microsoft IIS Site: `http://www.iis.net/`
[47] `http://d2rq.org/`
[48] Apache Tomcat Web site: `http://tomcat.apache.org/`

most important M2M transformation rules specified in S^{m}4RIA in order to help designers to create the mock-up of some models and thus speed up the development of SRIAs.

Table 7.11 introduces the five main model-to-model transformations designed for the S^{m}4RIA process, which can be specifically invoked during the processes of designed performed in the first or the second S^{m}4RIA activities. All the transformations are optional. They were created to help designers and simplify repetitive tasks but they should decide if the transformations can actually help the design depending on the requirements of the application. All the transformations introduced are unidirectional, i.e., the transformation cannot be reversed due to the information lost during the process.

Table 7.11. Summary of the Model-to-Model transformations rules in S^{m}4RIA.

Model-to-Model Transformation rules	Origin Model(s)	Resulting Model(s)
Domain2EDM	Domain Model	Extended Domain Model
EDM2ENM	Extended Domain Model	Extended Navigation Model
Pres&Orch2Visu	Extended Presentation Model Extended Orchestration Model	Visualisation Ontology Model
Domain2Navigation	Domain Model	(Extended) Navigation Model
Navigation2Presentation	(Extended) Navigation Model	(Extended) Presentation Model (Extended) Orchestration Model

The first three transformations (*Domain2EDM*, *EDM2ENM* and *Pres&Orch2Visu*) are part of the S^{m}4RIA core and can directly help to reduce the time for developing a SRIA. The two last transformation rules (*Domain2Navigation* and *Navigation2Presentation*) can be used in processes of modernisation and generation of interfaces with a low degree of variability, such as those for visualising raw data, e.g., in administrative applications. The following paragraphs briefly describe each of these model-to-model transformations.

The *Domain2EDM* transformation rule creates a mock-up of the domain ontology of the application (represented as the Extended Domain Model) from the information of the data structures included in the Domain model. From this initial model, the designer can import external ontologies or knowledge sources in order to create the final ontology.

The *EDM2ENM* transformation rule generates the view of the Extended Navigational Model for Semantic Web agents based on the knowledge captured in the domain ontology of the Extended Domain Model. The model view generated should be refined by the designer in order to filter the access to the ontology instances.

The *Pres&Orch2Visu* transformation rule generates the Visualisation Ontology Model from the information of the Extended Presentation and Orchestration model. In this case, given that the resulting model should not be manipulated by the designers, this transformation should be used in any case to avoid inconsistencies in the resulting ontologies of the SRIA.

The *Domain2Navigation* transformation can be used to generate a mock-up of an (Extended) Navigation model from the entities of the Domain model. This is a generic transformation that creates all the possible navigation paths given the domain classes, associations and class operations. The definition of the navigation through the data is an aspect of the application that is strongly dependent on the requirements of the application. However, there exist some auxiliary interfaces, such as the administrator interface, in which the data navigation and the operations invoked are similar in all the applications.

The *Navigation2Presentation* transformation generates a default interface from the information contained in the (Extended) Navigation models generated with the *Domain2Navigation* transformation. The definition of user interfaces is also a process very dependent on the requirements of the stakeholders. However, it is possible to define a fixed set of widgets given a pattern of navigation for some types of interfaces, such as those for administrators. This transformation facilitates the creation of this type of user interfaces, in which users will be able to manage the application data with no restrictions.

The following subsections are focused on explaining the transformations of the first group, i.e., the *Domain2EDM*, *EDM2ENM* and *Pres&Orch2Visu* transformation rules, including the code of the actual transformation. The transformations for software modernisation can be found in Section F.2 (page 287).

In this case, the transformations were implemented using QVT operational as rule language, which is a variant of the QVT language that allows designers to define transformations using imperative (explicitly invoked) instead of declarative rules (invoked after a certain condition is fulfilled).

7.3.1 Model-to-Model Transformations to Obtain the Extended Domain Model

The first example of model-to-model transformation is the *Domain2EDM* transformation, which, as mentioned, before creates a mock-up of the EDM. The *Domain2EDM* rule employs a collection of sub-rules that address the transformation of the different elements of the model. Table 7.12 shows a list of the transformation sub-rules, the Domain model elements employed as input of the transformation and the resulting EDM elements.

Regarding the input elements from the Domain model, the *ConceptualModel* element can be used to represent the whole model or a sub-package, which modifies the output of the transformation. In the same way, the Attribute objects can be used for representing class attributes or association roles, thus yielding different results.

Table 7.12. Summary of the input and output model elements in the Domain2EDM transformation.

Transformation Rule	Element in the Origin Model	Resulting Element(s)
Model2Model	ConceptualModel (root)	EDModel + OntologyModel
ConceptualModel2OntologyModel	ConceptualModel (leaf)	OntologyModel
Class2Concept	Class	Concept
Attribute2Attribute	Attribute (class attribute)	Attribute
Attribute2ObjectProperty	Attribute (association role)	ObjectProperty
Association2Association	Association	Association
Inheritante2Inheritance	Inheritance	Inheritance

Table 7.13 introduces the different rules defined for the Domain2EDM rule expressed using QVT operational.

Table 7.13. Code of the Domain2EDM QVTo model-to-model transformation rule.

```
modeltype ConceptualView uses "http://www.insidesoft.net/conceptualView/1.0.0";
modeltype EDM uses "http://www.dlsi.ua.es/ooh/sm4ria/edm/1.0";
```

```
transformation Domain2EDM(in inModel: ConceptualView, out outModel: EDM);

main()
{
    inModel.rootObjects()[ConceptualModel]->map Model2Model();
}
```

```
mapping ConceptualModel::Model2Model() : EDModel
{
    result.name := "default";

    //result.models += self.map CModel2OModel();
    self.map ConceptualModel2OntologyModel(result, null);

    var localModel := result.models[OntologyModel]->selectOne(isLocal = true);

    // Create default Source element
    var source := object Source
    {
        name := localModel.name + "-source";
        uriBase := "http://default.source.com";
        type := SourceType::SPARQL;
    };

    result.models += source;

    result.relations += object Instance
    {
        id := "instance";
        base := localModel;
        target := source;
    };
}
```

```
mapping ConceptualModel::ConceptualModel2OntologyModel(inout edModel : EDModel, in
lastOModel : OntologyModel)
{
    var model = object OntologyModel {};

    model.name := self.name;
    model.isLocal := true;
    model.namespace := self.name.toLower();
    model.uriBase := "http://" + model.namespace + ".com";
    model.description := "New model";
    model.conceptualModel := self;

    model.elements += self.elements[Class]->map Class2Concept( );
    model.elements += self.elements[Association]->map Association2Association(
model );
    model.elements += self.elements[Inheritance]->map Inheritance2Inheritance(
model );

    edModel.models += model;
    if (lastOModel <> null ) then
    {
        edModel.relations += new OntoImport(lastOModel.name + "_" + model.name,
lastOModel, model);
    }
    endif;

    self.elements[ConceptualModel]->map ConceptualModel2OntologyModel( edModel,
model );
}

constructor edm::OntoImport::OntoImport (i : String, b: OntologyModel, t :
OntologyModel)
{
    id := i;
    base := b;
    target := t;
}
```

```
mapping Class::Class2Concept() : Concept
{
    result.name := self.name;
    result.uri := self.name.toLower();
    result.description := "New class '" + self.name + "'";
    result.domainClass := self;

    result.properties += self.attributes[associationOrigin = null and
associationTarget = null]->map Attribute2Attribute();
    result.properties += self.attributes[navigable = true and (associationOrigin <>
null or associationTarget <> null)]->map Attribute2ObjectProperty();
}
```

```
mapping conceptualView::Attribute::Attribute2Attribute() : edm::Attribute
{
    result.name := self.name;
    result.uri := self.name.toLower();
    result.domainAttribute := self;

    result.target := self.type.PType2XmlType();
}

mapping conceptualView::Attribute::Attribute2ObjectProperty() : edm::ObjectProperty
{
    result.name := self.name;
    result.uri := self.name.toLower();
    result.domainAttribute := self;
}
```

```
mapping conceptualView::Association::Association2Association(in model:
OntologyModel) : LinkedAssociation
{
    init
    {
        var concepts := model.elements[Concept];
        var objProperties := concepts.properties[ObjectProperty];
    }

    result.name := result.uri := "assoc_" + self.name.toLower();
    result.domainAssociation := self;

    result.conceptOrigin := concepts->select(e | e.domainClass = self.classOrigin)-
>first();
    result.conceptTarget := concepts->select(e | e.domainClass = self.classTarget)-
>first();

    result.direct := objProperties->select(p | self.rolOrigin.navigable and
p.domainAttribute = self.rolOrigin)->first();
    result.inverse := objProperties->select(p | self.rolTarget.navigable and
p.domainAttribute = self.rolTarget)->first();
}
```

```
mapping conceptualView::Inheritance::Inheritance2Inheritance(in model:
OntologyModel) : edm::Inheritance
{
    init
    {
        var concepts := model.elements[Concept];
    }

    result.uri := result.name := self.name.toLower();
    result.conceptualInheritance := self;

    result.ascendant := concepts->select(c | c.domainClass = self.father)->first();
    result.descendant := concepts->select(c | c.domainClass = self.son)->first();
}
```

This transformation generates a domain ontology with the elements of the local ontology in a new EDM. Subsequently, designers need to complete adding external ontologies and knowledge sources if necessary.

7.3.2 Model-to-Model Transformations to Obtain the Extended Navigation Model

The *EDM2ENM* transformation creates a mock-up of the view of the Extended Navigational Model for Semantic Web agents that designers should refine before generating the application. The *EDM2ENM* rule employs a collection of sub-rules that address the transformation of the different elements of the model. Table 7.14 shows a list of the transformation sub-rules, the EDM elements used as input of the transformation and the resulting ENM elements.

Table 7.14. Summary of the input and output model elements in the EDM2ENM transformation.

Transformation rules	Element in the Origin Model	Resulting Element(s)
Model2Model	EDModel (root)	ENModel (root)
OntologyModel2ENModel	OntologyModel (isLocal = true)	ENModel
Concept2ENClass	Concept	ExtNavigationalClass
Attribute2NavAttribute	Attribute	ExternalNavigationalAttribute
Association2Link	ObjectProperty	ExternalTraversalLink
Association2Link	Association	--
--	Inheritance	--

Table 7.15 introduces the different rules defined for the EDM2ENM rule expressed using QVT operational.

Table 7.15. Code of the EDM2ENM QVTo model-to-model transformation rule

```
modeltype EDM uses "http://www.dlsi.ua.es/ooh/sm4ria/edm/1.0";
modeltype ENM uses "http://www.dlsi.ua.es/ooh/sm4ria/enm/1.0";
modeltype NAV uses "http://www.insidesoft.net/navigationalView";
```

```
transformation EDM2ENM(in inModel : EDM, out outModel : ENM);

main()
{
    inModel.rootObjects()[EDModel].models[isLocal = true].map Model2Model();
}
```

```
mapping OntologyModel::Model2Model() : ENModel when { self.isLocal = true }
{
    result.edModel := self.edModel;
    result.name := self.name;

    // Create "home" class
    var home := object ExtNavigationalClass
        {
            name := "home";
            isEntryPoint := true;
        };

    result.navigationalElem += home;
    result.navigationalElem += self.elements[Concept]->map Concept2ENClass();
    self.elements[Association]->map Association2Link( result );

    var localSource := result.edModel[Instance]->selectOne(base.isLocal =
true).target;

    // Create the links between the home class and the rest
    result.navigationalElem[ExtNavigationalClass]->forEach(elem)
    {
        result.navigationalElem += object ExternalTraversalLink
        {
            name :=  "home-" + elem.name;
            nodeOrigin := home;
            nodeTarget := elem;
            activationMode := ActivationType::Manual;
            source := localSource;
        }
    };
}
```

```
mapping Concept::Concept2ENClass() : ExtNavigationalClass
{
    result.name := self.name;
    result.edmConcept := self;
    result.referToClass := self.domainClass;
    result.isEntryPoint := false;
    result.navAttribute += self.properties[Attribute]->map
Attribute2NavAttribute();
}
```

```
mapping EDM::Attribute::Attribute2NavAttribute() : ExternalNavigationalAttribute
{
    result.name := self.name;
    result.edmAttribute := self;
    result.referToAttribute := self.domainAttribute;
}
```

```
mapping Association::Association2Link(inout model : ENModel)
{
    var extNavClasses := model.navigationalElem[ExtNavigationalClass];
    var origin := extNavClasses->selectOne(edmConcept = self.conceptOrigin);
    var target := extNavClasses->selectOne(edmConcept = self.conceptTarget);
    var localSource := model.edModel.relations[Instance]->selectOne(base.isLocal =
true).target;
```

```
    if (self.direct.domainAttribute.navigable)
    then
        model.navigationalElem += object ExternalTraversalLink
        {
            name := self.name + "-" + self.direct.name;
            nodeOrigin := origin;
            nodeTarget := target;
            activationMode := ActivationType::Manual;
            source := localSource;
            associationRol := self.direct.domainAttribute;
            edmObjectProperty := self.direct;
        }
    endif;

    if (self.inverse.domainAttribute.navigable)
    then
        model.navigationalElem += object ExternalTraversalLink
        {
            name := self.name + "-" + self.inverse.name;
            nodeOrigin := target;
            nodeTarget := origin;
            activationMode := ActivationType::Manual;
            source := localSource;
            associationRol := self.inverse.domainAttribute;
            edmObjectProperty := self.inverse;
        }
    endif;
}
```

After the transformation process, the designer will need to refine the model introducing constraints that limit the sharing of the data with the external agents.

7.3.3 MODEL-2-MODEL TRANSFORMATIONS TO OBTAIN THE VISUALISATION ONTOLOGY MODEL

Despite the fact that the VOM can be created directly from scratch, the S^{m}4RIA process defines a model-to-model transformation called *Pres&Orch2Visu*, which can automatically generate the VOM from the Extended Presentation and Orchestration models.

In the same manner that the previous transformations, the *Pres&Orch2Visu* rule employs a collection of sub-rules that address the transformation of the different elements of the model, shown in Table 7.16 with the elements used as input of the transformation and the resulting VOM elements.

Table 7.16. Summary of the input and output model elements in the Pres&Orch2Visu transformation.

Transformation Rule	Element in the Origin Model	Resulting Element(s)
Model2Model	Presentation Model + Orchestration Model	VisualisationOntologyModel
ScreenShot2ScreenShot	ScreenShot	ScreenShot Contains
Widget2VE	Widget (Extended Presentation Model)	VisualElement
Annotation2Annotation	Annotation (EPM)	Annotation
Method2Action	WMethod (EPM)	Action RunnableAction
Param2ActParam	WMethodParameter (EPM)	ActionParameters
Event2Event	WEvent (EPM)	Event AvailableEvent
Call2Run	EventCall (Extended Orchestration Model)	Run

The specification of the sub-rules using the QVT operational rule language is introduced in Table 7.17.

Table 7.17. Code of the Pres&Orch2Visu QVTo model-to-model transformation rule.

```
mapping SLPresentationModel::Model2Model(in orchModel : EOM) : VisualisationModel
{
    result.name := self.name;
    result.elements += self.sshot->map Screenshot2Screenshot(result);
    result.epm := self;
    result.eom := orchModel;
}
```

```
mapping ScreenShot::Screenshot2Screenshot(inout model:VisualisationModel) :
VOM::ScreenShot
{
    result.name := self.name;

    result.uri := "default uri";

    self.referredWidgets->map Widget2VE(model)->forEach(ve)
    {
         result.se += ve;
    }

    result.ss := self;
}
```

```
mapping Widget::Widget2VE(inout model:VisualisationModel) : VisualElement
{
    init
    {
        result.name := self.name;
    }
}

mapping SLWidget::SLWidget2VE(inout model:VisualisationModel) : VisualElement
inherits Widget::Widget2VE
{
    init
    {
        if (self.oclIsTypeOf(SLButton)) then
        {
            result := object VOM::Button{}
        }
        // Check input type to return the correct type of element
        else
        {
            result := object VOM::SimpleElement()
        }
        endif;
    }

    // Copy aesthetic properties of the widget

    result.annotations += self.annotations->map Annotation2Annotation();

    result.widget := self;

    self.methods->map WMethod2Action()->forEach(a)
    {
        model.elements += a;

        model.elements += object RunnableAction
        {

        }
    };

    self.events->map WEvent2Event()->forEach(e)
    {
        model.elements += e;
        model.elements += object AvailableEvent
           {
              component := result;
              event := e;
           }
    };

    model.elements += self.ecas->Call2Run(model);

}
```

```
mapping Annotation::Annotation2Annotation() : VOM::Annotation
{
    result.uri := self.uri;
    result.refAnnotation := self;
}
```

```
mapping WEvent::WEvent2Event() : Event
{
    result.name := self.name;

    // Mapping WEventParameter elements contained in self.parameters
}
```

```
mapping WMethod::WMethod2Action() : VOM::Action
{
    result.name := self.name;

    // Mapping WMethodParameter elements contained in self.parameters
}
```

```
mapping EventCall::Call2Run(in model : VisualisationModel) : VOM::Run
{
    result.event := model.elements->select(e | e.oclIsTypeOf(VOM::Event) and e.name
= self.event.name)->asSequence()->first().oclAsType(VOM::Event);

    result.action := model.elements->select(a | a.oclIsTypeOf(VOM::Action) and
a.name = self.conditions.trueActions->first().name)->asSequence()->
first().oclAsType(VOM::Action);
}
```

The resulting model of this transformation can be used in the model-to-text transformations processes with no adaptation or update.

The three examples explained offer a general view of the model-to-model transformations in the S^{m}4RIA process, which completes the description of the methodology. The next section describes the CASE tool that supports the development processes of the S^{m}4RIA methodology, including a collection of model editors and the rule engines for the invocation of the transformation rules explained in this section.

7.4 CONCLUSIONS

This chapter concluded the description of the S^{m}4RIA methodology with the explanation of the model-to-model and model-to-text transformation rules involved in the development process. In order to define these transformations, a reference architecture for SRIAs was proposed in the first subsection using the experience gained from the manual development of the case studies. Subsequently, a set of model-to-text transformation rules were defined that address the generation of the architectural components and resources of the SRIA. These transformations were specified using the Xpand language, which expresses the transformation rules as a collection of code templates.

In order to speed up the design processes of a SRIA, the S^{m}4RIA methodology includes a collection of model-to-model transformation rules, which help designers to create model mock-ups or even complete models. The purpose of these rules is to reduce the time and effort that developers spend in repetitive tasks when creating new models. The defined transformations could be classified in two groups: a) the transformations oriented to accelerate the core tasks of the S^{m}4RIA process and b) the ones that address processes of modernisation based on the S^{m}4RIA process.

The previous subsections introduced the main details of the transformations, as well as their code (Xpand or QVTo) and an example of result in some cases. These transformations were successfully tested using the proposed case studies. Notwithstanding this, a further analysis of the benefits and shortcomings of the architecture proposed and the transformations could be carried out using a group of real developers, who could assess the resulting applications based on their professional experience.

Chapter 8. IMPLEMENTATION OF THE METHODOLOGY: S^{M}4RIA EXTENSION FOR OIDE

In order to assess the S^{m}4RIA methodology and facilitate its adoption, their models and transformation processes were implemented as an extension of the OIDE tool[49] called *S^{m}4RIA Extension for OIDE* (Hermida et al., 2012a, 2012b). This tool implements the S^{m}4RIA models and automates the transformation processes (model-to-model and model-to-text, explained in previous subsections) needed for generating SRIAs.

This tool was also developed to validate the components of the S^{m}4RIA methodology by modelling and developing the use cases proposed in Sections 3.4 and 3.5.2 (pages 53 and 60). In a first stage, the development of the tool was used for detecting those possible lacks or drawbacks of the method. Once the development process was in its final stages, the tool was externally evaluated in two forums (national and international), in which the opinions held by the experts were taken into consideration to refine the method and the tool.

The tool also implements the main mechanisms of modernisation described in the Sm4RIA-M configuration in order to allow developers to generate rich user interfaces from ontologies and to automatically generate administration views for the designed applications.

[49] OOH4RIA Integrated Development Environment (Meliá et al., 2010b).

8.1 OOH4RIA Integrated Development Environment

OIDE is an application based on the Eclipse framework, developed as a set of Eclipse plug-ins, which supports the OOH4RIA methodology for the development of RIAs. Specifically, this application defines the OOH4RIA meta-models using the EMOF/Ecore format and, using the EMF[50]/GMF[51] framework, facilitates the definition using a graphical concrete syntax of the OOH4RIA models: Domain, Navigational and Presentation-Orchestration. In OIDE, Presentation and Orchestration models are integrated into a single model, i.e., the OIDE Presentation Model, developed using the GMF framework. The OOH4RIA Orchestration model is represented as a new section in the Properties window.

Furthermore, OIDE supports the generation processes that obtain most of the RIA software components (both server and client modules). The model-to-text generation rules are implemented as a set of Xpand rules, which, at present, transform the information contained in the models into C# code contained into a Visual Studio solution. The transformation rules use the WCF and the NHibernate frameworks for the development of the server modules and the Silverlight framework for the user interface. At present, the generation rules of OIDE are being adapted to the generation of HTML5 Rich Internet Applications, whose server modules are developed in Java technologies and the client modules using HTML5 and JavaScript.

OIDE does not implement any transformation process between OOH4RIA models but contains a collection of wizards and helpers that assist users in the process of creating models and elements and generating the RIA application.

Figure 8.1 illustrates a screenshot of the OIDE user interface showing an example of an OIDE Presentation Model. The interface is divided in four areas, inherited from the Eclipse IDE:

- *Model editors:* the central area of interface shows the different model editors of the IDE and the Palette, which is the tool bar that shows the buttons for creating the elements of model.

[50] Eclipse Modeling Framework: `http://www.eclipse.org/modeling/emf/`
[51] Graphical Modeling Project: `http://www.eclipse.org/modeling/gmp/`

- *Project explorer:* the left-side area shows the different projects and the folder and file structure.
- *Property bar:* the down-side area (below the Model Editor area) shows the properties of the elements and the outline of the models represented.

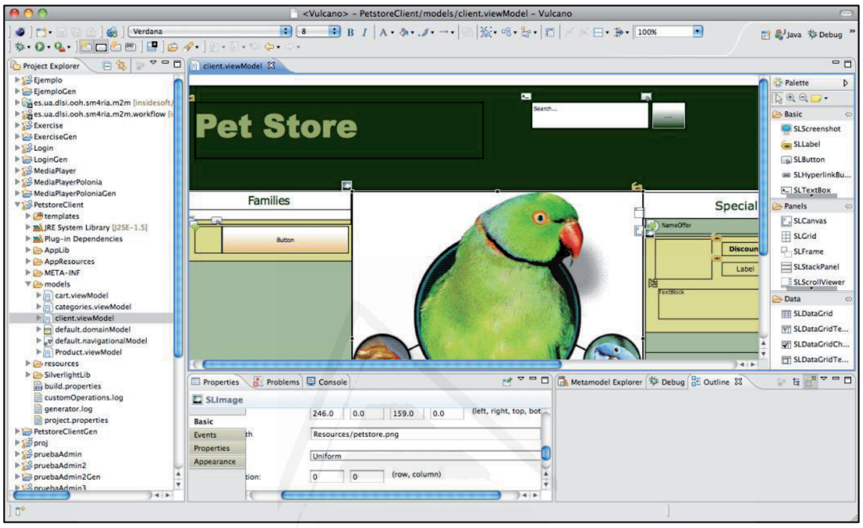


Figure 8.1. Screenshot of the main interface of the OIDE tool.

8.2 Models and Transformations

Using OIDE as platform, the S^{m}4RIA extension for OIDE implements the artefacts and processes of the S^{m}4RIA methodology as a new functionality of Eclipse. This section describes the elements developed and the modifications to the original tool that facilitate the design of the SRIA software components.

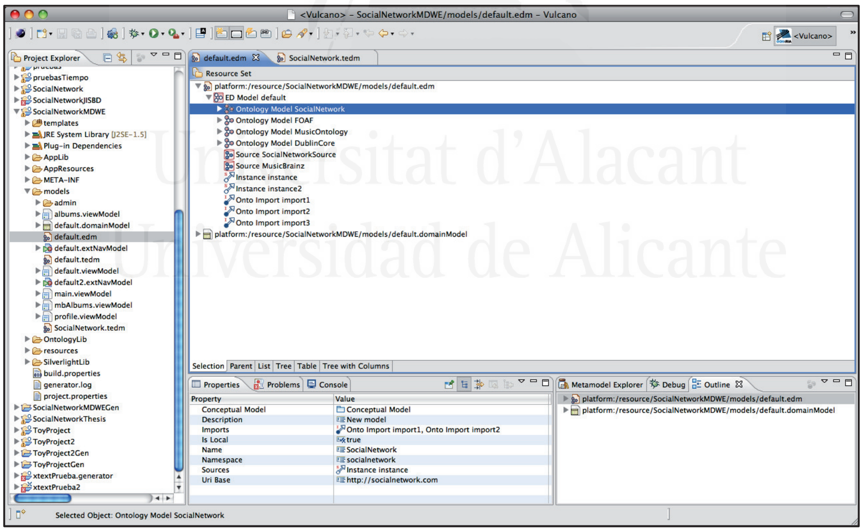
8.2.1 Model Editors

This extension implements the editors of the S^{m}4RIA models, the transformation rules and the workflows that manage the generation of the SRIA applications using the frameworks provided by Eclipse (e.g., EMF, GMF, Xtext[52], Xpand, QVT operational or MWE[53]). More

[52] Xtext Web site: `http://www.eclipse.org/Xtext/`
[53] Modeling Workflow Engine:
`http://wiki.eclipse.org/Modeling_Workflow_Engine_%28MWE%29`

specifically, two new model editors have been implemented from scratch, i.e., the Extended Domain Model and the Visualisation Ontology Model; and two models have been extended from the existing OOH4RIA implementation: the Extended Navigation Model, the Extended OIDE Presentation Model. Furthermore, new wizards and helpers have been developed in order to help users to create the elements of the models. The models included in this tool can be described as follows:

- **Extended Domain Model.** In order to create the editor for this model, its metamodel was specified using the EMOF/Ecore meta-metamodel included in the Eclipse EMF framework. From the metamodel, this framework also generates all the elements of a tree-based, editor in which designers can create new models, and the API for managing the elements of the model from the transformation processes. A screenshot of the EMF editor in OIDE is illustrated in Figure 8.2, showing the example of the SNS case study (see Section 5.2.2, page 94).



```
pruebasTiempo
SocialNetwork
SocialNetworkJISBD
SocialNetworkMDWE
  templates
  JRE System Library [J2SE-1.5]
  Plug-in Dependencies
  AppLib
  AppResources
  META-INF
  models
    admin
    albums.viewModel
    default.domainModel
    default.edm
    default.extNavModel
    default.tedm
    default.viewModel
    default2.extNavModel
    main.viewModel
    mbAlbums.viewModel
    profile.viewModel
    SocialNetwork.tedm
  OntologyLib
  resources
  SilverlightLib
  build.properties
  generator.log
  project.properties
SocialNetworkMDWEGen
SocialNetworkThesis
ToyProject
ToyProject2
ToyProject2Gen
ToyProjectGen
xtextPrueba.generator
xtextPrueba2
```

```
platform:/resource/SocialNetworkMDWE/models/default.edm
  ED Model default
    Ontology Model SocialNetwork
    Ontology Model FOAF
    Ontology Model MusicOntology
    Ontology Model DublinCore
    Source SocialNetworkSource
    Source MusicBrainz
    Instance instance
    Instance instance2
    Onto Import import1
    Onto Import import2
    Onto Import import3
platform:/resource/SocialNetworkMDWE/models/default.domainModel
```

Property	Value
Conceptual Model	Conceptual Model
Description	New model
Imports	Onto Import import1, Onto Import import2
Is Local	true
Name	SocialNetwork
Namespace	socialnetwork
Sources	Instance instance
Uri Base	http://socialnetwork.com

```
platform:/resource/SocialNetworkMDWE/models/default.edm
platform:/resource/SocialNetworkMDWE/models/default.domainModel
```

Selected Object: Ontology Model SocialNetwork

Figure 8.2. Screenshot of the S^m4RIA extension for OIDE showing the EMF representation of the Extended Domain Model for the SNS case study.

In order to facilitate the creation of the EDM and due to the limitations of the EMF editors, an alternative Xtext editor was developed (see Figure 8.3), which allows the specification of the model using a textual notation. The syntax was firstly generated from the Extended Domain Model and then refined to improve the usability of the resulting language. The final syntax of the Extended Domain Model language is defined in Section G.3 (page 303).

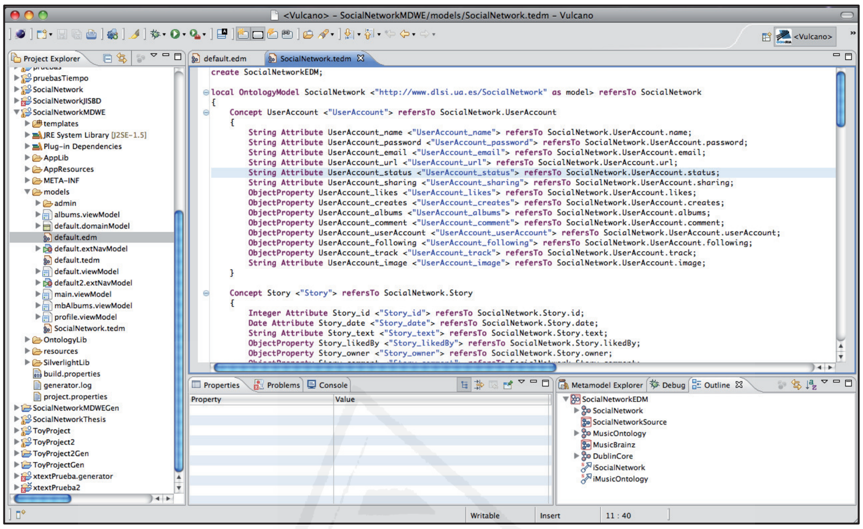
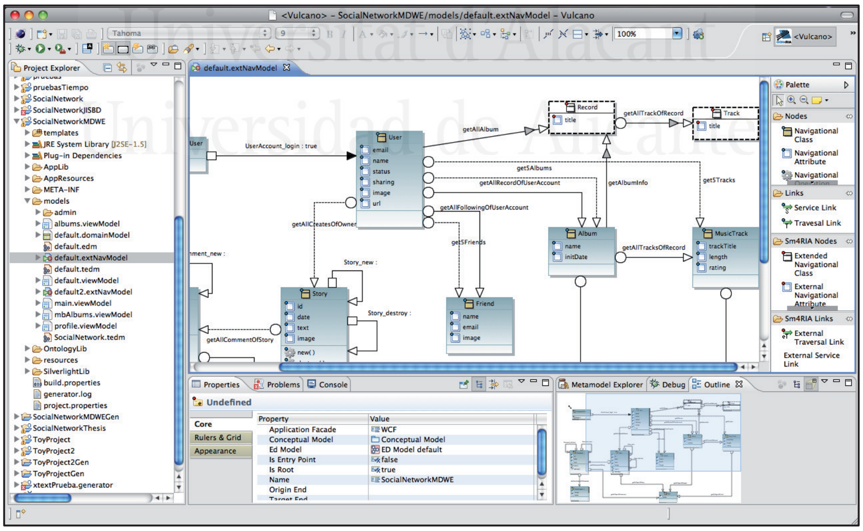

```
create SocialNetworkEDM;

local OntologyModel SocialNetwork <"http://www.dlsi.ua.es/SocialNetwork" as model> refersTo SocialNetwork
{
    Concept UserAccount <"UserAccount"> refersTo SocialNetwork.UserAccount
    {
        String Attribute UserAccount_name <"UserAccount_name"> refersTo SocialNetwork.UserAccount.name;
        String Attribute UserAccount_password <"UserAccount_password"> refersTo SocialNetwork.UserAccount.password;
        String Attribute UserAccount_email <"UserAccount_email"> refersTo SocialNetwork.UserAccount.email;
        String Attribute UserAccount_url <"UserAccount_url"> refersTo SocialNetwork.UserAccount.url;
        String Attribute UserAccount_status <"UserAccount_status"> refersTo SocialNetwork.UserAccount.status;
        String Attribute UserAccount_sharing <"UserAccount_sharing"> refersTo SocialNetwork.UserAccount.sharing;
        ObjectProperty UserAccount_likes <"UserAccount_likes"> refersTo SocialNetwork.UserAccount.likes;
        ObjectProperty UserAccount_creates <"UserAccount_creates"> refersTo SocialNetwork.UserAccount.creates;
        ObjectProperty UserAccount_albums <"UserAccount_albums"> refersTo SocialNetwork.UserAccount.albums;
        ObjectProperty UserAccount_comment <"UserAccount_comment"> refersTo SocialNetwork.UserAccount.comment;
        ObjectProperty UserAccount_userAccount <"UserAccount_userAccount"> refersTo SocialNetwork.UserAccount.userAccount;
        ObjectProperty UserAccount_following <"UserAccount_following"> refersTo SocialNetwork.UserAccount.following;
        ObjectProperty UserAccount_track <"UserAccount_track"> refersTo SocialNetwork.UserAccount.track;
        String Attribute UserAccount_image <"UserAccount_image"> refersTo SocialNetwork.UserAccount.image;
    }

    Concept Story <"Story"> refersTo SocialNetwork.Story
    {
        Integer Attribute Story_id <"Story_id"> refersTo SocialNetwork.Story.id;
        Date Attribute Story_date <"Story_date"> refersTo SocialNetwork.Story.date;
        String Attribute Story_text <"Story_text"> refersTo SocialNetwork.Story.text;
        ObjectProperty Story_likedBy <"Story_likedBy"> refersTo SocialNetwork.Story.likedBy;
        ObjectProperty Story_owner <"Story_owner"> refersTo SocialNetwork.Story.owner;
```

Figure 8.3. Screenshot of the Sm4RIA extension for OIDE showing the Xtext representation of the Extended Domain Model for the SNS case study.



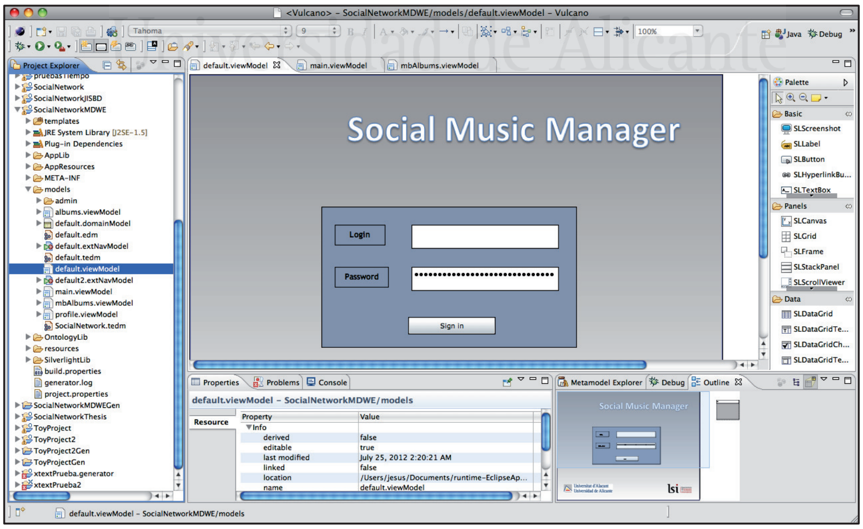
Undefined

Property	Value
Application Facade	WCF
Conceptual Model	Conceptual Model
Ed Model	ED Model default
Is Entry Point	false
Is Root	true
Name	SocialNetworkMDWE
Origin End	

Figure 8.4. Screenshot of the Sm4RIA extension for OIDE showing the GMF representation of the Extended Navigational Model for the SNS case study.

- **Extended Navigational Model**. The editor of this model extends the GMF editor of the OOH4RIA Navigation model including the tools for defining new external navigational classes from the EDM and external navigational links, which could be combined creating data/knowledge mashups. At present, the tool helps to access the main Linked Data services, i.e., the SPARQL endpoints. Figure 8.4 depicts the screenshot of the tool showing this editor with the example for the SNS case study (see Section 5.3.2, page 107).
- **Extended OIDE Presentation Model.** The editor for the Extended Presentation and Orchestration models extends the one for the OIDE Presentation model with new properties for including semantic annotations and establishing relationships between the elements of the Extended Presentation and Orchestration models and the ones of the Extended Navigation Model.

  The following figures (Figure 8.5, Figure 8.6 and Figure 8.7) show three screenshots of the Extended Presentation and Orchestration models for the SNS case study, modelled in the OIDE platform (corresponding to the models introduced in Section 6.1.2, page 126).

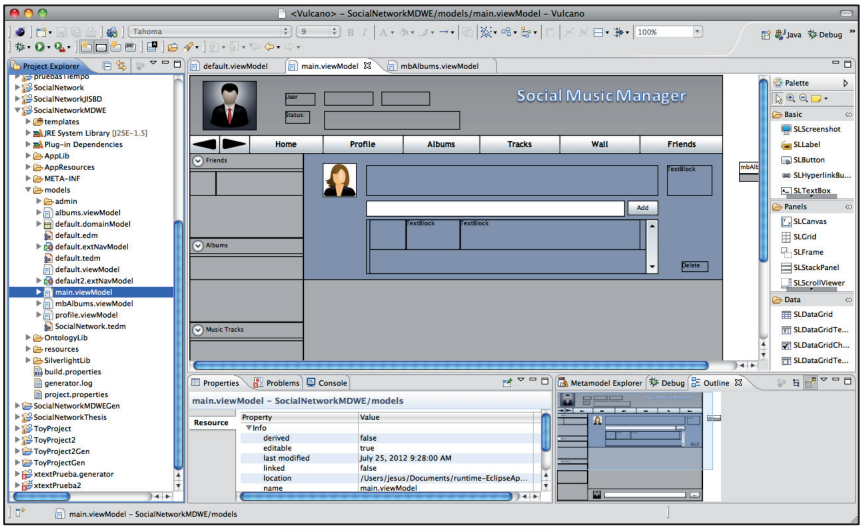


default.viewModel – SocialNetworkMDWE/models

Resource

Property	Value
Info	
derived	false
editable	true
last modified	July 25, 2012 2:20:21 AM
linked	false
location	/Users/jesus/Documents/runtime-EclipseAp...
name	default.viewModel

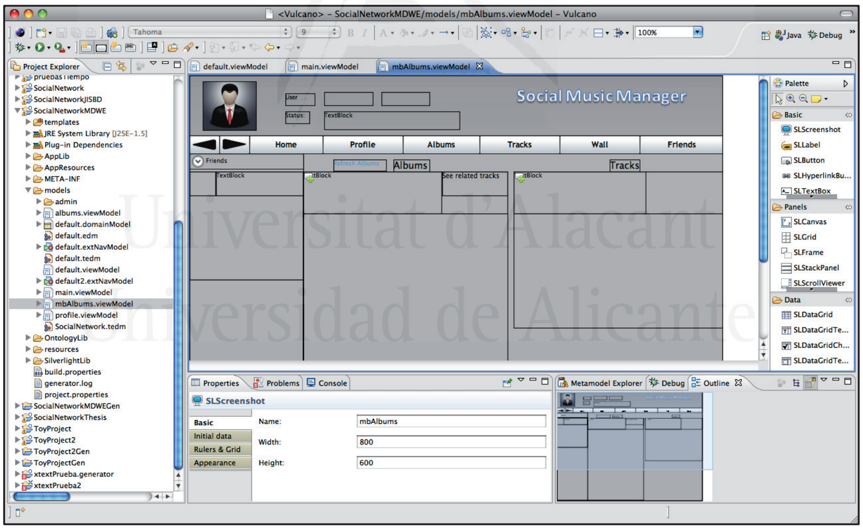
Figure 8.5. Screenshot of the S^{m}4RIA extension for OIDE showing the GMF representation of the OIDE Presentation Model for the SNS case study (Default screenshot).



main.viewModel – SocialNetworkMDWE/models

Property	Value
Info	
derived	false
editable	true
last modified	July 25, 2012 9:28:00 AM
linked	false
location	/Users/jesus/Documents/runtime-EclipseAp...
name	main.viewModel

Figure 8.6. Screenshot of the Sm4RIA extension for OIDE showing the GMF representation of the OIDE Presentation Model for the SNS case study (Main screenshot).



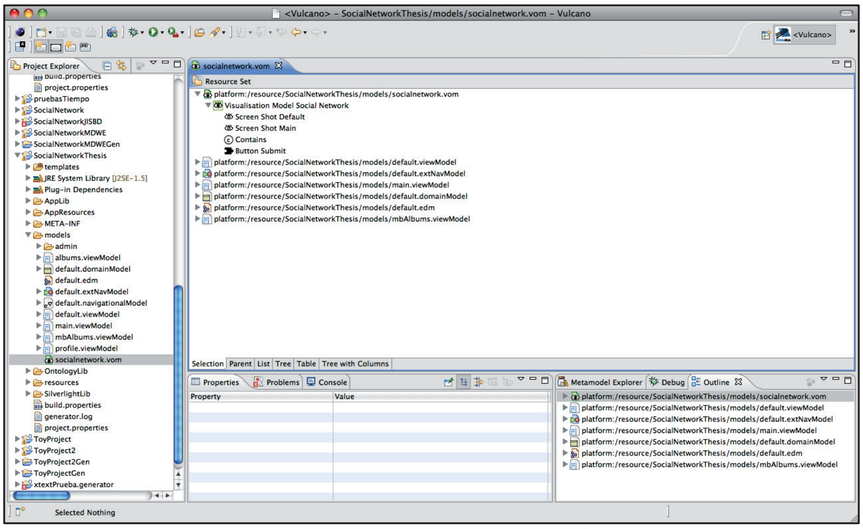
SLScreenshot

Basic | Initial data | Rulers & Grid | Appearance

Name: mbAlbums
Width: 800
Height: 600

Figure 8.7. Screenshot of the Sm4RIA extension for OIDE showing the GMF representation of the OIDE Presentation Model for the SNS case study (Main screenshot)

- **Visualisation Ontology Model**. Following the same process that with the EDM editor, the editor for the Visualisation Ontology Model was implemented based on its EMF/Ecore metamodel following the same process that the EDM editor.


<Vulcano> – SocialNetworkThesis/models/socialnetwork.vom – Vulcano

Project Explorer

- build.properties
- project.properties
- pruebasTiempo
- SocialNetwork
- SocialNetworkJISBD
- SocialNetworkMDWE
- SocialNetworkMDWEGen
- SocialNetworkThesis
  - templates
  - JRE System Library [J2SE-1.5]
  - Plug-in Dependencies
  - AppLib
  - AppResources
  - META-INF
  - models
    - admin
    - albums.viewModel
    - default.domainModel
    - default.edm
    - default.extNavModel
    - default.navigationalModel
    - default.viewModel
    - main.viewModel
    - mbAlbums.viewModel
    - profile.viewModel
    - socialnetwork.vom
  - OntologyLib
  - resources
  - SilverlightLib
  - build.properties
  - generator.log
  - project.properties
- ToyProject
- ToyProject2
- ToyProject2Gen
- ToyProjectGen
- xtextPrueba.generator

socialnetwork.vom

Resource Set

- platform:/resource/SocialNetworkThesis/models/socialnetwork.vom
  - Visualisation Model Social Network
    - Screen Shot Default
    - Screen Shot Main
    - Contains
    - Button Submit
- platform:/resource/SocialNetworkThesis/models/default.viewModel
- platform:/resource/SocialNetworkThesis/models/default.extNavModel
- platform:/resource/SocialNetworkThesis/models/main.viewModel
- platform:/resource/SocialNetworkThesis/models/default.domainModel
- platform:/resource/SocialNetworkThesis/models/default.edm
- platform:/resource/SocialNetworkThesis/models/mbAlbums.viewModel

Selection | Parent | List | Tree | Table | Tree with Columns

Properties | Problems | Console

Property	Value

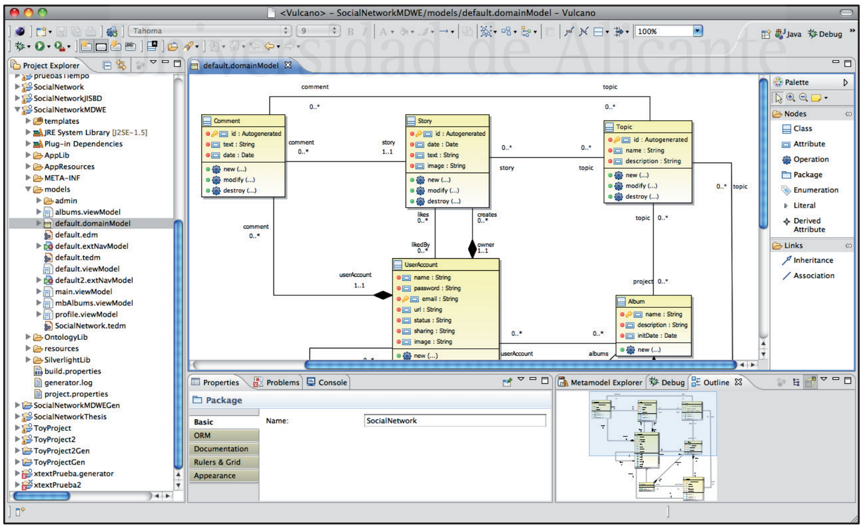
Metamodel Explorer | Debug | Outline

- platform:/resource/SocialNetworkThesis/models/socialnetwork.vom
- platform:/resource/SocialNetworkThesis/models/default.viewModel
- platform:/resource/SocialNetworkThesis/models/default.extNavModel
- platform:/resource/SocialNetworkThesis/models/main.viewModel
- platform:/resource/SocialNetworkThesis/models/default.domainModel
- platform:/resource/SocialNetworkThesis/models/default.edm
- platform:/resource/SocialNetworkThesis/models/mbAlbums.viewModel

Selected Nothing

Figure 8.8. Screenshot of the S^{m}4RIA extension for OIDE showing the EMF view of the Visualisation Ontology Model for the SNS case study.

- **OOH4RIA Domain Model**. The editor of the Domain Model was directly reused from the implementation included in OIDE. To complete the description of the model editors available in the tool, Figure 8.9 depicts a screenshot of the OIDE tool showing the Domain Model editor with the example of the SNS case study (see Section 5.1.2, page 87).



<Vulcano> – SocialNetworkMDWE/models/default.domainModel – Vulcano

Project Explorer

- pruebasTiempo
- SocialNetwork
- SocialNetworkJISBD
- SocialNetworkMDWE
  - templates
  - JRE System Library [J2SE-1.5]
  - Plug-in Dependencies
  - AppLib
  - AppResources
  - META-INF
  - models
    - admin
    - albums.viewModel
    - default.domainModel
    - default.edm
    - default.extNavModel
    - default.tedm
    - default.viewModel
    - default2.extNavModel
    - main.viewModel
    - mbAlbums.viewModel
    - profile.viewModel
    - SocialNetwork.tedm
  - OntologyLib
  - resources
  - SilverlightLib
  - build.properties
  - generator.log
  - project.properties
- SocialNetworkMDWEGen
- SocialNetworkThesis
- ToyProject
- ToyProject2
- ToyProject2Gen
- ToyProjectGen
- xtextPrueba.generator
- xtextPrueba2

default.domainModel

Properties | Problems | Console

Package

Basic | ORM | Documentation | Rulers & Grid | Appearance

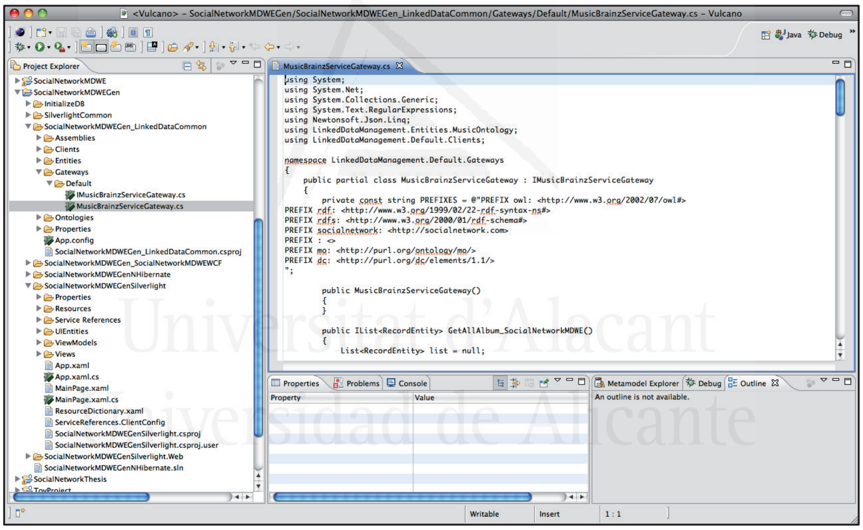
Name: SocialNetwork

Figure 8.9. Screenshot of the OIDE tool showing the GMF representation of the Domain Model for the SNS case study.

8.2.2 Transformation Processes

The S^{m}4RIA extension for OIDE implements all the transformation rules explained in the previous sections of this chapter:

- **Model-to-Text transformations.** In order to generate the software components specific for SRIAs, this extension includes the Xpand transformation rules of the S^{m}4RIA third activity and executes them using the information obtained from the model editors and the Eclipse Xpand transformation engine. Figure 8.10 shows a screenshot of the application with the Visual Studio project generated from the SNS case study and an editor with the code of the *MusicBrainzServiceGateway* component.



<Vulcano> – SocialNetworkMDWEGen/SocialNetworkMDWEGen_LinkedDataCommon/Gateways/Default/MusicBrainzServiceGateway.cs – Vulcano
Java
Debug
Project Explorer
SocialNetworkMDWE
SocialNetworkMDWEGen
InitializeDB
SilverlightCommon
SocialNetworkMDWEGen_LinkedDataCommon
Assemblies
Clients
Entities
Gateways
Default
IMusicBrainzServiceGateway.cs
MusicBrainzServiceGateway.cs
Ontologies
Properties
App.config
SocialNetworkMDWEGen_LinkedDataCommon.csproj
SocialNetworkMDWEGen_SocialNetworkMDWEWCF
SocialNetworkMDWEGenNHibernate
SocialNetworkMDWEGenSilverlight
Properties
Resources
Service References
UIEntities
ViewModels
Views
App.xaml
App.xaml.cs
MainPage.xaml
MainPage.xaml.cs
ResourceDictionary.xaml
ServiceReferences.ClientConfig
SocialNetworkMDWEGenSilverlight.csproj
SocialNetworkMDWEGenSilverlight.csproj.user
SocialNetworkMDWEGenSilverlight.Web
SocialNetworkMDWEGenNHibernate.sln
SocialNetworkThesis
MusicBrainzServiceGateway.cs
Properties
Problems
Console
Property
Value
Metamodel Explorer
Debug
Outline
An outline is not available.
Writable
Insert
1 : 1

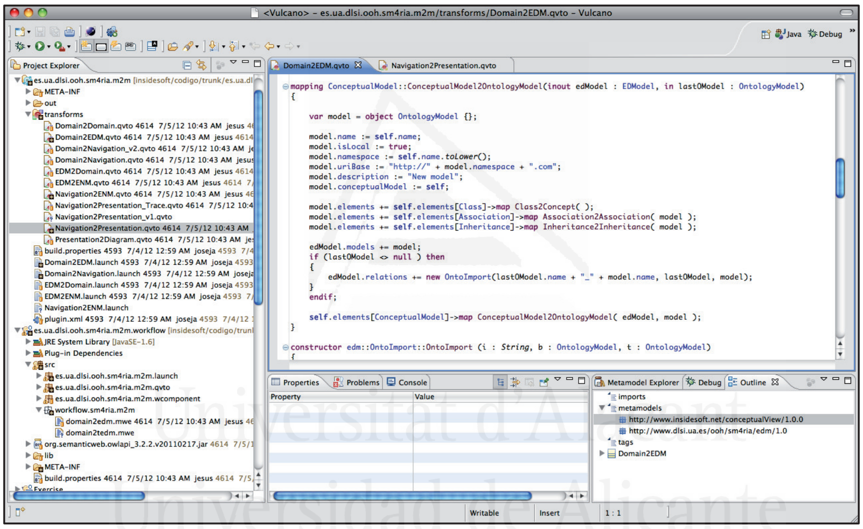
```
using System;
using System.Net;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using Newtonsoft.Json.Linq;
using LinkedDataManagement.Entities.MusicOntology;
using LinkedDataManagement.Default.Clients;

namespace LinkedDataManagement.Default.Gateways
{
    public partial class MusicBrainzServiceGateway : IMusicBrainzServiceGateway
    {
        private const string PREFIXES = @"PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX socialnetwork: <http://socialnetwork.com>
PREFIX : <>
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
";

        public MusicBrainzServiceGateway()
        {
        }

        public IList<RecordEntity> GetAllAlbum_SocialNetworkMDWE()
        {
            List<RecordEntity> list = null;
```

Figure 8.10. Screenshot of the S^{m}4RIA extension for OIDE showing the Visual Studio project resulting from the model-to-text transformation processes for the SNS case study.

- **Model-to-Model transformations.** Apart from the wizards and helpers included in the model, the tool includes a collection of M2M rules that facilitate creation of new models from existing ones. The tool extension implements the QVT operational rules introduced in the last section and invokes them using the Eclipse QVT Operational rule engine. Specifically, the transformations defined in this extension are the following (Ma – Mb transformations are unidirectional, i.e., they transform model Ma into Mb):

- *Domain2EDM* transformation: Domain model – Extended Domain Model.
- *EDM2Domain* transformation (beta): Domain model – Extended Domain Model.
- *EDM2ENM* transformation: Extended Domain Model – Extended Navigation Model.
- *Navigation2Presentation* transformation: Extended Navigation Model – Presentation Model.

Figure 8.11 illustrates a screenshot of the tool with the QVTo projects developed and an example of transformation implemented in the Eclipse QVTo editor.



```
es.ua.dlsi.ooh.sm4ria.m2m [insidesoft/codigo/trunk/es.ua.dl
  META-INF
  out
  transforms
    Domain2Domain.qvto 4614  7/5/12 10:43 AM  jesus 46
    Domain2EDM.qvto 4614  7/5/12 10:43 AM  jesus 4614
    Domain2Navigation_v2.qvto 4614  7/5/12 10:43 AM  je
    Domain2Navigation.qvto 4614  7/5/12 10:43 AM  jesus
    EDM2Domain.qvto 4614  7/5/12 10:43 AM  jesus 4614
    EDM2ENM.qvto 4614  7/5/12 10:43 AM  jesus 4614  7/
    Navigation2ENM.qvto 4614  7/5/12 10:43 AM  jesus 46
    Navigation2Presentation_Trace.qvto 4614  7/5/12 10:4
    Navigation2Presentation_v1.qvto
    Navigation2Presentation.qvto 4614  7/5/12 10:43 AM
    Presentation2Diagram.qvto 4614  7/5/12 10:43 AM  je
  build.properties 4593  7/4/12 12:59 AM  joseja 4593  7/4
  Domain2EDM.launch 4593  7/4/12 12:59 AM  joseja 4593
  Domain2Navigation.launch 4593  7/4/12 12:59 AM  joseja
  EDM2Domain.launch 4593  7/4/12 12:59 AM  joseja 4593
  EDM2ENM.launch 4593  7/4/12 12:59 AM  joseja 4593  7/
  Navigation2ENM.launch
  plugin.xml 4593  7/4/12 12:59 AM  joseja 4593  7/4/12
es.ua.dlsi.ooh.sm4ria.m2m.workflow [insidesoft/codigo/trun
  JRE System Library [JavaSE-1.6]
  Plug-in Dependencies
  src
    es.ua.dlsi.ooh.sm4ria.m2m.launch
    es.ua.dlsi.ooh.sm4ria.m2m.qvto
    es.ua.dlsi.ooh.sm4ria.m2m.wcomponent
    workflow.sm4ria.m2m
      domain2edm.mwe 4614  7/5/12 10:43 AM  jesus 46
      domain2tedm.mwe
  org.semanticweb.owlapi_3.2.2.v20110217.jar 4614  7/5/1
  lib
  META-INF
  build.properties 4614  7/5/12 10:43 AM  jesus 4614  7/5,
Exercise
```

```
mapping ConceptualModel::ConceptualModel2OntologyModel(inout edModel : EDModel, in lastOModel : OntologyModel)
{

    var model = object OntologyModel {};

    model.name := self.name;
    model.isLocal := true;
    model.namespace := self.name.toLower();
    model.uriBase := "http://" + model.namespace + ".com";
    model.description := "New model";
    model.conceptualModel := self;

    model.elements += self.elements[Class]->map Class2Concept( );
    model.elements += self.elements[Association]->map Association2Association( model );
    model.elements += self.elements[Inheritance]->map Inheritance2Inheritance( model );

    edModel.models += model;
    if (lastOModel <> null ) then
    {
        edModel.relations += new OntoImport(lastOModel.name + "_" + model.name, lastOModel, model);
    }
    endif;

    self.elements[ConceptualModel]->map ConceptualModel2OntologyModel( edModel, model );
}

constructor edm::OntoImport::OntoImport (i : String, b : OntologyModel, t : OntologyModel)
{
```

```
imports
metamodels
  http://www.insidesoft.net/conceptualView/1.0.0
  http://www.dlsi.ua.es/ooh/sm4ria/edm/1.0
tags
Domain2EDM
```

Figure 8.11. Screenshot of the Sm4RIA extension for OIDE showing the M2M projects developed and the Domain2EDM model-to-model transformation in the Eclipse QVTo editor.

8.2.3 New Processes of Software Modernisation

The new artefacts introduced in the last two subsections facilitate the adaptation of the Sm4RIA methodology to new processes of modernisation and generation as described in Section 4.2.5.2, page 76. At present, these processes are under testing in the *Sm4RIA extension for OIDE*. The most relevant processes of modernisation are the following:

- *Automatic generation of administrator views for applications.* Using the M2M transformations already implemented, it is possible to automatically generate UIs for SRIA administrators (or facilitate the generation of most of their modules) from the Sm4RIA EDM or the OOH4RIA Domain model. Figure 8.12 and Figure 8.13 show a screenshot of the tool with the Navigation and Presentation models automatically generated for the SNS case study from the Domain model. At present, the tool generates a default presentation model. Still, it is necessary to study different possibilities of personalisation of the resulting interface depending on the designer's preference.

- *Generation of RIA interfaces for Linked Data sources*. By means of two new transformations that obtain a Domain model and an EDM from an OWL ontology, it is possible to specify a RIA server that manage the data of a Linked Data service and subsequently define a RIA client that visualize them. This process has only been assessed with simple ontologies.

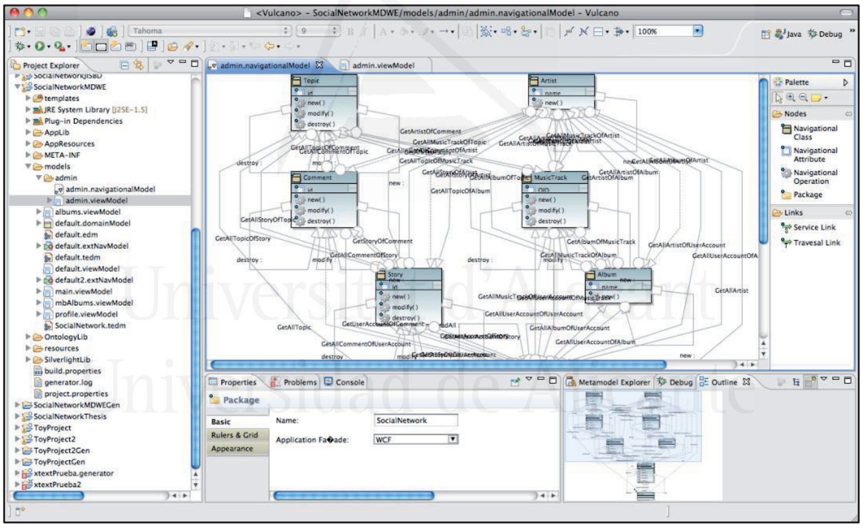
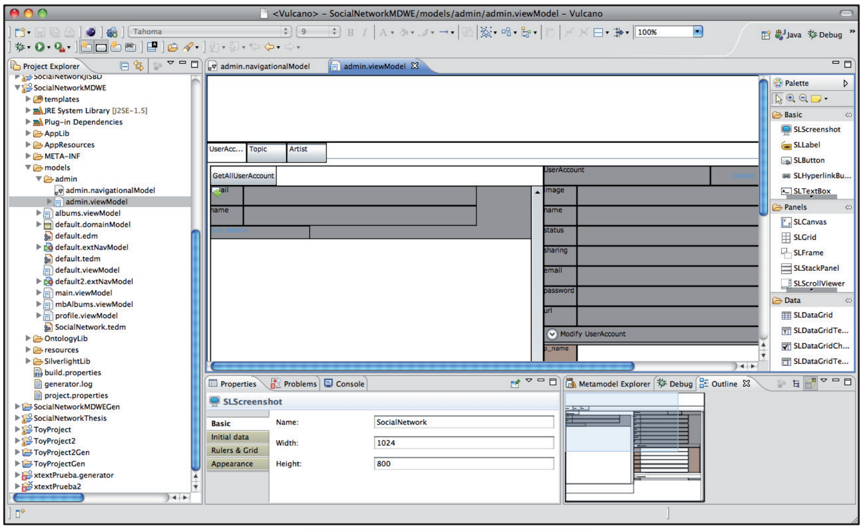


Figure 8.12. Screenshot of the S^{m}4RIA extension for OIDE showing the Navigational model for the administration view of the SNS case study.



SLScreenshot

Basic	Name:	SocialNetwork
Initial data	Width:	1024
Rulers & Grid	Height:	800
Appearance		

Figure 8.13. Screenshot of the S^{m}4RIA extension for OIDE showing the Presentation model for the administration view of the SNS case study.

8.3 CONCLUSIONS

The final part of the chapter introduces the *S^{m}4RIA extension for OIDE*, i.e., an application that implements the S^{m}4RIA process using the OIDE CASE tool as a basis. The development of this application was used to assess the methodology in two stages. The first one was an interative process of validation, in which, the development of the tool was employed to improve the S^{m}4RIA modelling artefacts. In a second stage, the tool was externally assessed in national and international forums, thus obtaining valuable comments from external experts. The final success in the development of this tool proves the viability of the approach.

This chapter has introduced the main features of the tool and a set of screenshots showing them. In order to clarify its contributions, the following three tables summarise the main components implemented in the tool. Table 8.1 shows the main model editors implemented and the concrete syntaxes supported for each of them. The tree-based editor is automatically generated by the EMF framework. The basic elements of the S^{m}4RIA model have already been implemented in the tool. The elements of the OIDE Presentation model for the S^{m}4RIA-B configuration are under development.

Table 8.1. Summary of the model editors included in the extension.

Editors	Implemented components			
	EMF Tree-based editor	Textual editor (Xtext)	Visual editor (GMF)	Include wizards and helpers
Domain Model Editor	Yes, from OIDE	Under development	Yes, from OIDE	Yes, from OIDE
Extended Domain Model editor	Yes	Yes	Under development	Yes, needs improvements in usability
Extended Navigational Model editor	Yes	No	Yes	Yes, needs improvements in usability
OIDE Extended Presentation Model editor	Yes	No	Yes. Under development the visualisation elements specific for the Business Intelligence field.	Yes, adapted from OIDE. Under development the elements in S^{m}4RIA-B.

Table 8.2 shows the main features of the two main sets of model-to-model transformations, i.e., for the generation of the SRIA server and client modules, respectively. The software generators currently generate SRIAs adapted to the .NET framework version 4.0 and the generation of Java and HTML SRIAs is under development. The tool generates D2RQ mapping rules for the D2RQ server, which can perform the transformation from data objects to ontology instances.

Table 8.2. Summary of the features of the code generator processes.

Feature	SRIA Server Generator	SRIA Client Generator
Generator language	Xpand 4, Xtend 1	Xpand 4, Xtend 1
Output languages	C# 4.0	C# 4.0
	Java 6 (under development)	HTML 4.1, JavaScript (under development)
Frameworks used	Nhibernate 3.1,Windows Communication Foundation	Silverlight 4
	Hibernate 3.1, AXIS2	JQuery
Architectural pattens (from Section 7.1)	Data Access Object, Data Transfer Object, Distributed Façade, Service Gateway, Service Interface	Model View View Model
Partial code generated	Yes	No
Generation of ontologies	XML OWL 1.1	--
Generation of database-to-RDF	D2RQ mapping rules	--
Generation of ontology instances	XML RDF	XML RDF

Finally, Table 8.3 summarises the QVTo model-to-model transformations implemented in the tool and the development process to which they are associated. This table includes the transformations needed in the processes of modernisation and automatic user interface generation described in Section 8.2.3.

Table 8.3. Summary of the model-to-model transformations supported by the tool.

Development process	Transformation	Input model	Output model	Status
S^m4RIA	Domain2EDM	Domain	EDM	Supported
	Nav2Pres	ENM	OIDE EPM	Supported for designing interfaces for adminitrators
	Nav&Pres2Orch			
	Pres&Orch2Visu	OIDE EPM	VOM	Supported. Under development elements specific for the Business Intelligence field.
S^m4RIA-M	Ontology2EDM (text-to-model)	OWL 1.1 ontology	EDM	Partially supported. Needs improvements in multiple, transitive ontology imports.
	EDM2Domain	EDM	Domain	Supported

The empirical evaluation of the non-functional requirements of the methodology and the CASE tool (e.g., usability, maintainability and easiness of use) is being conducted for OOH4RIA and S^m4RIA by means of a set of experiments. The study of the non-functional properties is being performed using statistical analysis over the experiences of a real group of developers, who test the models of the methodologies using real designs based on their background. The first experiments have been already performed over the OOH4RIA/S^m4RIA Domain model, assessing its maintainability, ease of use and the overall impression of the developers. The results were positive as shown by Martinez et al. (Martinez et al., 2013)

This second analysis complements the functional analysis carried out in this thesis, ensuring that the methodology and the tool fulfil the requirements for the development of SRIA, and can help to detect new aspects to improve the methodology. Moreover, these aspects can determine the final degree of adoption of the tool in real business scenarios, and thus, the actual success of the approach.

Chapter 9. Conclusions & Future Work

The last chapters described in detail the main aspects of the SRIA proposal and the S^{m}4RIA methodology, which are the two main contributions of this thesis. Each of the chapters contained a specific section which drew partial conclusions about the topics addressed. Using a different approach, the purpose of this last chapter is to highlight the general contributions of this thesis, their benefits and limitations based on the problems detected in the first two chapters. More specifically, the first part of the chapter revisits the research questions stated in Chapter 1 and offers an answer to each of them based on the content of the core chapters of the thesis. From the analysis of the limitations found and the conclusions obtained in the rest of the chapters, the second part describes the main lines of future work.

9.1 Conclusions

Based on the findings throughtout this thesis, this section aims to answer the three research questions introduced in Chapter 1, focusing on the contributions brought by the present work, their main benefits and limitations. The following paragraphs will answer each of the questions:

RQ1 – *Is it possible to improve the interoperability of Rich Internet Applications with other software systems (such as, Web search engines) using existing techniques, technologies and resources from the Semantic Web?*

This research question was affirmatively answered with the proposal of Semantic Rich Internet Application introduced in Chapter 3, which is the first contribution of this thesis. SRIAs are designed as an extension of traditional RIAs that employ Semantic Web technologies in order to represent the knowledge they use and share it across the Web. The use of the Linked data principles, which are a de-facto standard supported by the W3C, and the technologies for representing and sharing knowledge, mainly ontologies represented in OWL, facilitates the reuse of knowledge published by other applications. Chapter 3 specifies a complete set of requirements for this new type of Semantic Web application and explains how to structure the knowledge of the application using three orthogonal ontologies: domain, navigation and visualisation, which can be used in complex processes of search. By changing the domain ontology of the application, developers could share knowledge in different domains. The information contained in the navigational ontology could guide the navigation of the users across the RIA. The third ontology can be used to share information of the multimedia elements included in the application.

This chapter also describes the basic structure of the application used as a reference in the thesis. This structure offers a general view of the application and emphasizes the differences between traditional RIA and SRIA. As could be appreciated, SRIAs include new software modules in their server for sharing ontology instances and, in those applications with HTML interfaces, embed semantic annotations directly within the content visualised by the browser. The description of this approach is completed in Chapter 7, where the architecture of the application is proposed. This chapter describes in detail the software components that should be developed per each software module of Chapter 3 and their relationship. In order to facilitate the development of the applications, Chapter 7 also showed the manner in which some of the functionalities could be externalised by using one of the existing database-to-RDF mapping tools, i.e., D2RQ.

The assessment process followed in Chapter 3 was based on the development of a collection of case studies, which demonstrated that *a)* the expected benefits regarding knowledge sharing can be achieved; and *b)* the data interoperability and visibility of the RIA data in some Web clients, such as Web searchers, can be improved. Using these predefined scenarios, the evaluation process ensures that the proposed requirements are fulfilled and the desired functionalities are included in the application, thus solving the issues detected in RIA. Moreover, these case studies were externally assessed in international conferences and journals in order to ensure the validity of the conclusions that were drawn (see Annex A).

In the case of the applications for business intelligence, this thesis proposed an adaptation of the original approach, i.e., the RI@BI approach. In order to deal with the special requirements of this type of applications, the proposed adaptation combine techniques for knowledge management and visualisations and business-to-business services.

The main limitation of the approach is that it does not consider non-functional requirements related to the general performance of the SRIA applications in real scenarios. Despite the fact that the SRIAs developed for the evaluation are similar to real applications, it would be also necessary to measure quantitative parameters of performance (e.g., response time, memory used, CPU used) under different load conditions and information queries to validate the software architecture proposed and the alternative D2R mapping tool. With the results of this empirical evaluation, it would be possible to establish different architectural configurations according to different non-functional requirements set by the stakeholders.

Another aspect of the proposal that has not assessed in this thesis is the benefit of combining the three ontologies (representing different knowledge) in the annotation model proposed in Section 3.2 (page 46). Although there are some previous approaches that use ontologies to represent domain data, navigational structures or user interface elements, the benefits of combining the three in order to improve the searches of information has not been empirically demonstrated. This assessment process requires the implementation of a specialised client that employs this data, which was out of the scope and objectives of this thesis.

RQ2 – *How can the existing model-driven methodologies be extended in order to develop the solution to the problems detected in Rich Internet Applications?*

In order to answer this question, Chapter 4 introduced the S^{m}4RIA methodology, which is the second contribution of this thesis, as an extension of the OOH4RIA methodology, for the development of SRIA.

As could be appreciated from the analysis in Section 2.2.3 (page 37), none of the existing model-driven Web engineering methodologies effectively combined the elements required for the development of this type of applications. Existing methodologies (e.g., WebML) contain part of the elements needed (development of RIA, ontologies, access to Web services), but these elements remain unconnected. In addition, they are not yet aligned to the new initiatives for knowledge management of the Semantic Web, such as the Linked Data approach, which facilitates the processes of knowledge publication and exploitation. Table 9.1 and Table 9.2 extend the analysis perfomed in the second chapter including and highlighting the features of S^{m}4RIA. S^{m}4RIA effectively combines the artefacts for modelling RIA (extending the ones included in OOH4RIA) and the required primitives for modelling the components for knowledge management and sharing with the purpose of developing SRIAs.

The methodology facilitates the design of the processes of sharing (and reusing) knowledge as linked data (following the Linked Data principles) in SRIAs. The simplicity of their processes and models should reduce the learning curve and enable non-expert designers to use Linked Data sources in order to import fresh knowledge from the Web to their applications. S^{m}4RIA also provides the elements required to represent the knowledge and services shared by a RIA using different techniques (e.g., SPARQL endpoints).

This thesis has already introduced two configurations of the S^{m}4RIA process: one for the development of RI@BI (*S^{m}4RIA-B*, Section 4.2.5.1, page 74) and another oriented to the modernisation of legacy applications and the generation of RIA interfaces to knowledge bases (*S^{m}4RIA-M*, Section 4.2.5.2, page 76). The first configuration groups the elements that designers need to develop of SRIA in the field of Business Intelligence, which includes new modelling elements for representing complex data visualisations and the access to B2B services.

Table 9.1. Comparison of S^{m}4RIA with other methodologies (RIA design features).

	Objective	Feature	Methodologies							
			S^{m}4RIA	**WebML**	**UWE**	**UWE-R**	**RUX-Method**	**OOHDM**	**OOWS**	**SHDM**
General		Type	**Model-driven**	Model-driven	Model-driven	Model-driven	Model-driven	Model-driven	Model-driven	Ontology-driven
General	O3	CASE tool	**OIDE**	WebRatio	ArgoUWE		RUX Tool			
Rich Internet Applications	O2	Develop Rich Internet Applications	**Yes**	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Rich Internet Applications		Type of methodo-logy (Busch & Koch 2009)	**a, extends OOH4RIA**	a	d	a	b	a	a, extends OO-Method	a
Rich Internet Applications	O1	Design RIA server	**Yes**	Yes	Yes	Yes	No	Yes	Yes	Yes
Rich Internet Applications	O1	Design the structure of Rich User Interfaces	**Yes, with the Extended Presentation Model**	Yes, with the Hypermedia model	Yes, with a new set of patterns over the Presentation model	Yes, extending the Presentation model	Yes, with the Concrete Interface	Yes, with Abstract Data Views	Yes, with the Interaction model	Yes, with the Presentation ontology
Rich Internet Applications	O1	Design the behaviour of Rich User Interfaces	**Yes, with the Extended Orchestration Model**	Yes, with the Hypermedia model	Yes, with a new set of patterns over the Navigation and the Process model	Yes, extending the Navigation and Process models	Yes, with the Concrete Interface	Yes, with ADV-charts	Yes, with the Interaction model	Yes, with the Presentation ontology
Rich Internet Applications	O1	Generate browser-oriented rich clients	**Yes**	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Rich Internet Applications	O1	Generate plugin-oriented rich clients	**Yes**	Yes	No	No	No	No	No	No
Rich Internet Applications	O3	CASE tool support	**S^{m}4RIA extension for OIDE**	WebRatio	ArgoUWE		RUX Tool			

Table 9.2. Comparison of S^m4RIA with other methodologies (Semantic Web application design features).

	Objective	Feature	Methodologies						
			S^{m}4RIA	WebML	RUX-Method	SHDM	WSDM	OntoWebber	Hera/Hera-S
Semantic Web Applications	O2	Develop Semantic Web Applications	**Yes**	Yes	Yes	Yes	Yes	Yes	Yes
	O1.1 O1.2 O1.3	Design ontologies	**Yes**	Yes, using the Ontology model	No	Yes, with the Domain ontology	Yes	Yes, with the Domain model	Yes, in the Data collection activity
	O1.3	Import ontologies	**Yes**	Yes	No, only works with OntoRUX.	Yes, with the domain ontology	Yes	Yes	Yes
	O1.3	Reuse external knowledge bases	**Yes**	Yes, with Semantic Web Services	No	Yes, in another approach (De Souza Bomfim & Schwabe, 2011)	No	No	
	O1.1 O1.2	Generate ontology instances	**Yes**	Yes	Yes	Yes	Yes	Yes	Yes
	O1.1 O1.2	Generate semantically annotated UIs	**Yes**	Yes	Yes	Yes	Yes	Yes	Yes
	O1.1	Generate a service for accessing the ontology instances	**Yes, Linked Data endpoints**	Yes, Semantic Web Services	No	No	No		Yes, based on SeRQL
	O1.1 O1.2 O1.3	Aligned to the Linked Data principles	**Yes**	No	No	Yes	No	No	No
	O1.1	Generate Linked Data datasets	**Yes**	No	No	No	No	No	No
	O1.3	Reuse Linked Data datasets	**Yes**	No	No	Yes, in another approach (De Souza Bomfim & Schwabe, 2011)		No	No
	O1	Design rich UIs	**Yes**	No	No	Yes	No	No	No
	O3	CASE tool support	**S^{m}4RIA extension for OIDE**	-	editRUX, included in RUXTool	Synth platform		OntoWebber	

The second configuration facilitates the creation of SRIAs from existing databases or ontologies by means of text-to-model and model-to-model transformations. In this configuration, developers can exploit the use of model-to-model transformation rules in order to reengineer traditional Web applications –and create new (S)RIAs– and simplify the design of those parts shared by many applications, e.g., interfaces for administrators. Further possibilities for the modernisation processes with S^{m}4RIA will be analysed and assessed more in detail in future works.

The validation of the models and the processes included in the methodology was driven by the design of different case studies using a SRIA as a platform, with different requirements in the same domain. The case study chosen to explain the features of the process from Chapter 5 to 7 was the social network site because of its complexity and requirements, and also because it is well-known in the field of Web Engineering.

The methodology has been applied to two ongoing projects, with public and private funding, that aim at developing Linked Data repositories with rich user interfaces using a SRIA as a platform.

The development of the case studies and the application to the development projects has shown the benefits of S^{m}4RIA:

(a) it facilitates the definition of most of the SRIA software modules at design time, offering an overview of the resulting application before it is actually generated, and protects the customised code introduced by the developers from regenerations;

(b) it reduces the cost in terms of time and resources involved in developing and maintaining SRIAs (which is also a consequence of the implementation in a CASE tool). Several of the main tasks of the methodology can be automated with the model transformations (model-to-model and model-to-text).

(c) it enables non-expert users to employ the knowledge bases available on the Web in their applications and to create new ones; and

(d) it simplifies the creation and exploitation of Linked Data services in RIAs.

Despite the known benefits, however, model-driven methodologies have not yet reached a high degree of adoption in business scenarios, mainly due to the mentality of the stakeholders, who are reluctant to using automatically generated code. This is due to the fact that such code might, on the one hand, not be entirely adapted to their necessities and, on the other hand, that it is obtained through model-to-text generation processes, which are usually developed as black boxes, so stakeholders might not trust the generated code.

Notwithstanding these practice-related issues, S^{m}4RIA is meant to help the development of SRIAs. As shown in Chapter 4, the S^{m}4RIA models specify most of the components of the application, and the generation processes are capable of generating the main structure of the application. Moreover, some parts can be manually completed by developers after the application has been generated, bearing no risk in case of code regeneration. Furthermore, the OOH4RIA Architectural model (Meliá et al., 2010a) can be adapted to the S^{m}4RIA methodology, thus adding to it mechanisms for the specification of the architectural variability of the resulting SRIA. It would also be possible to create new generation rules that, reusing existing models, enable the generation of different front-ends for other types of devices, e.g., mobile devices.

Empirically analysing the complexity of a model-driven methodology can be a challenge. Nevertheless, such an assessment is sometimes required so as to ensure that the learning curve does not grow steeply for users with a strong background in Web development. To this aim, some experiments have already been conducted over the OOH4RIA and S^{m}4RIA models yielding positive results (Martinez et al., 2013). These first experiments assessed the satisfaction of use and maintainability of the Domain model on groups of real Web developers. The goal is to replicate this evaluation process with the rest of the OOH4RIA/S^{m}4RIA models.

RQ3 – *How can the proposed solutions be implemented in a CASE tool?*

This third research question was addressed together with the second one. In this dissertation, after the presentation of the methodology, Chapter 8 introduced the main features of the CASE tool that implements the S^{m}4RIA methodology, called *S^{m}4RIA extension for OIDE*, thus answering this last question. As its name indicates, this tool extends the OIDE CASE tool, which implements the OOH4RIA methodology, with the new model editors and generation processes included in S^{m}4RIA. The reuse of the OIDE tool and the Eclipse modelling framework, on which OIDE is based, facilitated the development of the models and transformation processes.

At present, after a decade of new approaches in Web engineering, it is a fact that the success of a model-driven development methodology is clearly bound to the existence of a CASE tool that implements the models and the processes of software generation. As mentioned in Chapter 8, the functionalities of the tool were iteratively validated together with the elements of the methodology.

One last evaluation that should be conducted is the application of the methodology and the tool in a set of real projects using developers with different expertise levels and backgrounds as test subjects. In this way, it would be possible to improve the development of the model editors taking into consideration other non-functional requirements such as their usability (which were out of the scope of this thesis). Another feasible option could be to release the tool as open source and, subsequently, to gather the opinions of developers and users of the tool, thus yielding a wider analysis of the benefits and disadvantages of the tool. This third evaluation should be performed together with the authors of the OIDE tool or their authorisation since OIDE is not open source at this moment.

This evaluation can be used to shorten the process of training needed in order to be able to employ the CASE tool correctly, which normally depends on different factors, e.g., complexity of the methodology, level of usability of the tool, background of the designer, etc.

As a summary of the contributions of this thesis, Table 9.3 associates the objectives introduced in Chapter 1, stated from the research questions, and the tasks performed to fulfil them.

Table 9.3. Summary of the contributions of this thesis associated to the objectives.

Research question	Objective	Description	Tasks Performed
RQ1	O1	Improve the interoperability of Rich Internet Applications with text-driven software systems on the Web (e.g., searchers).	Development of a proposal of Semantic Rich Internet Application.
	O1.1	Improve the exportability of the data contained in Rich Internet Applications	• SRIAs include new server modules for sharing the information as linked data (e.g., knowledge base or the SPARQL service), which provides a standard manner of publishing the information. • SRIAs with HTML interfaces can also embed semantic annotations based on the annotation model proposed.
	O1.2	Improve the access to information related to multimedia elements.	The domain and the visualisation ontologies of the annotation model proposed can be used to share information of the multimedia elements by means of the SRIA software modules.
	O1.3	Reuse techniques, technologies and resources already developed in the Semantic Web	• The SRIA proposal reuses part of the Semantic Web architecture and the Linked data principles to share knowledge across the Internet. • Use of ontologies to represent the knowledge managed by SRIAs. • Use of the standard languages OWL, RDF and SPARQL for the representation of the ontologies, ontology instances and queries.
	O1.4	Develop a collection of use cases that assess the validity of the solution proposed.	• Development of a media player as a SRIA. • Development of a social network site as a SRIA. • Development of a social network site for enterprises as a RI@BI.
RQ2	O2	Design a model-driven methodology for the development of the solution.	Development of the S^{m}4RIA model-driven methodology for the development of SRIA.
	O2.1	Facilitate the development of the solution proposed in O1.	Design of new models in S^{m}4RIA adapted to the new features included in SRIA: • The Extended Domain Model, for the design of the domain ontology. • The Extended Navigation Model,

Research question	Objective	Description	Tasks Performed
			for specifying the manner in which the ontology instances are used in the application. • The Extended Presentation and Orchestration models, for the visualisation of the ontology instances imported from external sources. Design of a collection of model-2-model transformation that speeds up the creation of model mock-ups for designers.
	O2.2	Improve the maintainability of the solution proposed in O1.	As a model-driven methodology, changes in the requirements of an application would only imply modifications in the models and regenerate the software code. The Xpand framework protects personalised code when invoking the transformation rules.
	O2.3	Extend an existing methodology for the development of RIA.	Design of S^{m}4RIA as an extension of the OOH4RIA methodology, specialised in the development of traditional RIAs.
RQ3	O3	Implement the elements of the methodology designed in a CASE tool.	Development of the CASE tool called *S^{m}4RIA extension for OIDE*. Evaluation of the tool in research forums.

All in all, this thesis has shown the manner in which the techniques and technologies from one trend of the Web, i.e., the Semantic Web (or the Web of Data), can be applied to the problems found in another, i.e., RIA, thus reusing the efforts spent during the last decade of development of the Internet. In order to deal with the challenge of developing applications that combine the technologies from the both trends in the context of the Web engineering, this thesis showed the benefits of using a model-driven methodology (together with a CASE tool) and the manner in which the modelling primitives for each trend can be used together, simplified by the extensive use of model transformations.

The contributions introduced in this thesis open new business opportunities by directly applying the solutions designed in real scenarios and also opens new lines of future research, which are described in the next section.

9.2 Future Work

The last section presented an overview of the contributions of this thesis, their main benefits and limitations. Consequently, also in correspondence to the partial conclusions obtained after each chapter, new lines of future research deriving from this thesis are described:

Related to the Semantic Web field, the following lines of work remain open:

- *Empirical assessment of the annotation model proposed using a Semantic Web search engine.* The annotation model proposed should be evaluated in a practical manner. To do so, first, it is necessary to establish a set of parameters, new or reused from other approaches, to measure the performance of the searches of information over the new SRIAs. Secondly, a large number of SRIA applications should be developed to obtain more detailed conclusions.
- *Extension of a Semantic Web client to exploit the annotation model proposed.* The current Semantic Web searchers, such as *Sindice* or *Watson*, retrieve and index the ontologies and annotations contained in Web sites. The annotation model described could help these searchers to retrieve the information they need in a more efficient manner and to discriminate it according to the structure and visualisation of the Web site, also described as ontology instances. In order to evaluate the model, as a second step, one of the existing Semantic Web clients should be extended.
- *Empirical evaluation of the chosen architecture for SRIA.* The SRIA architecture was defined based on previous experiences in the development of RIA and Semantic Web applications. However, the SRIA architecture was proposed according to the desired functionalities and the issues found in RIA, ignoring the aspects related to non-functional requirements of the solution, such as the final performance of the applications. An exhaustive study of different types of SRIA should be performed using different

technologies and mapping tools, which could lead to improvements in the architecture proposed. This study should evaluate the performance of the resulting application at different load rates and queries of information.

- *Repeat the evaluation processes for SRIA with RI@BI.*
- *Automatic generation of mobile interfaces.* Using the information shared by means of the SRIA Linked Data service, it would be possible to automatically create user interfaces for mobile devices that interpret the ontology instances shared. This could be a solution to the lack of support of plugin-oriented RIAs in mobile devices, which are one of the main gateways to the Internet for users.

Related to the Web engineering field, the lines of research proposed are the following:

- *Empirical evaluation of the S^{m}4RIA methodology.* In order to detect new limitations of the methodology and facilitate the adoption in business scenarios, it is necessary to continue with the empirical assessment of the process and the models with a group of real developers, following the first experiments with the OOH4RIA Domain model (Martinez et al., 2013) that statistically evaluated its maintainability, ease of use and the general impression of the developers.
- *Adapt the OOH4RIA Architecture model to S^{m}4RIA.* The proposed methodology does not take into consideration the possible variations in the SRIA software architecture (either for clients or servers). The application of the OOH4RIA Architecture model (Meliá et al., 2010a) to S^{m}4RIA would facilitate the representation of the architectural variability of SRIAs.
- *Study the modernisation processes for the generation of interfaces.* S^{m}4RIA has shown the manner in which it is possible to generate interfaces to Linked Data sources. However, this approach can be further studied focusing on the manner in which data can be visualised (structure, behaviour and aesthetics) and the adaptation of the model-to-model transformations. Another aspect to analyse is the possibility to generate more complex interfaces, not only those for administrators.

- *Study the modernisation processes for the generation of mobile applications.* In the same way, it could be possible to define new modernisation processes in S^{m}4RIA that generate mobile applications that access the data exposed by SRIA, or Web user interfaces for visualising these data. From a more generic perspective, the application of techniques of Web personalisation to S^{m}4RIA could be studied.
- *Study the definition of product lines for the generation of SRIA.* The definition of product lines for the generation of SRIA would facilitate the personalisation of the process of generation. Developers would choose the SRIA desired from a prefixed set of options (e.g., type of SRIA, output technology, etc.), which control the activation of the model-to-text transformation rules invoked during the process. This approach has already been studied in OOH4RIA with RIA development (Meliá et al., 2010a).
- *Study the advantages and limitations of the models with textual notation against the ones with graphic notation.* The CASE tool implemented a textual concrete syntax of the EDM given that, in the Eclipse platform, the time of development for an Xtext textual editor is lower than a graphical one, which facilitates the creation of models mock-ups and the testing of model transformations. After the development of an editor with a graphical concrete syntax, it would be possible to empirically assess the benefits of each one for the modelling of the knowledge of domain.

Finally, there are some issues with the tool that could be analysed:

- *Complete the implementation of the model-to-text transformation rules for RI@BI.* The transformations for the generation of the RI@BI interfaces should be included in the CASE tool.
- *Complete the implementation of the modernisation processes of S^{m}4RIA.* The current implementation of the tool already includes most of the transformation rules defined by the S^{m}4RIA processes. However, the integration between the transformations and the model editors could be improved.
- *Improve the general usability of the tool.* The development of the CASE tool addressed the implementation of the main elements of the methodology and the process. However, there are other aspects related to the usability of the tool that should be

considered if the tool is aimed at real developers. Among them, the management of the S^{m}4RIA projects and the invocation of the model-to-text rules could be improved.

- *Improve the usability of the model editors.* Model editors are a relevant part of the CASE tool. They include all the functionalities required to model the S^{m}4RIA models. However, it is necessary to study those functionalities of the editor that can boost the efficiency of the developers when modelling. Moreover, the integration between the model editors and the transformation processes should be also improved.

REFERENCES

Acerbis, R., Bongio, A., Brambilla, M., Butti, S., 2007. WebRatio 5: an eclipse-based CASE tool for engineering web applications, in: Proceedings of the 7th International Conference on Web Engineering, ICWE'07. Springer-Verlag, Berlin, Heidelberg, pp. 501–505.

Alur, D., Malks, D., Crupi, J., 2003. Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall.

Allaire, J., 2002. Macromedia Flash MX - A next-generation rich client. Macromedia.

Allemang, D., 2010. Semantic Web and the Linked Data Enterprise Linking Enterprise Data, in: Wood, D. (Ed.), Linking Enterprise Data. Springer US, Boston, MA, pp. 3–23.

Barjis, J., Gupta, A., Sharda, R., 2011. Knowledge work and communication challenges in networked enterprises. Information Systems Frontiers 13, 615–619.

Barrasa Rodríguez, J., 2007. Modelo para la definición de correspondencias entre ontologías y modelos relacionales (PhD Thesis). Universidad Politécnica de Madrid, Madrid, Spain.

Benjamin, K., 2010. A Strategy for Efficient Crawling of Rich Internet Applications (MSc Thesis). University of Otawa, Otawa, Canada.

Benjamin, K., Bochmann, G. von, Dincturk, M.E., Jourdan, G.-V., Onut, I.-V., 2011. A Strategy for Efficient Crawling of Rich Internet Applications, in: Web Engineering - 11th International Conference, ICWE 2011, Paphos, Cyprus, June 20-24, 2011, Lecture Notes in Computer Science. Presented at the 11th

International Conference on Web Engineering (ICWE 2011), Springer, Paphos, Cyprus, pp. 74–89.

Berners-Lee, T., 1998a. Semantic Web Road map. World Wide Web Consortium. Available from: http://www.w3.org/DesignIssues/Semantic.html

Berners-Lee, T., 1998b. Relational Databases on the Semantic Web. World Wide Web Consortium. Available from: http://www.w3.org/DesignIssues/RDB-RDF.html

Berners-Lee, T., 2000. Semantic Web - XML 2000. World Wide Web Consortium.

Berners-Lee, T., 2006. Linked Data - Design Issues. World Wide Web Consortium. Available from: http://www.w3.org/DesignIssues/LinkedData.html

Berners-Lee, T., Hendler, J., Lassila, O., 2001. The Semantic Web. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American 284, 34–43.

Bettencourt, N., Maio, P., Pongó, A., Silva, N., Rocha, J., 2006. A Systematization and Clarification of Semantic Web Annotation Terminology, in: Proceedings of the International Conference on Knowledge And Decision Technologies (ICKEDS'06). pp. 27–34.

Bizer, C., Seaborne, A., 2004. D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs.

Bozzon, A., Comai, S., Fraternali, P., 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. ACM, New York, NY, USA, pp. 353–360.

Brambilla, M., Ceri, S., Facca, F.M., Celino, I., Cerizza, D., Cefriel, E. della V., 2007. Model-driven design and development of semantic Web service applications. ACM Trans. Internet Technol. 8, 3.

Brambilla, M., Facca, F.M., 2007. Building Semantic Web Portals with WebML. Lecture Notes in Computer Science 4607, 312–327.

Brambilla, M., Preciado, J.C., Linaje, M., Sánchez-Figueroa, F., 2008. Business Process-Based Conceptual Design of Rich Internet

Applications, in: Proceedings of the 2008 Eighth International Conference on Web Engineering. Presented at the ICWE '08, IEEE Computer Society, New York, USA, pp. 155–161.

Breslin, J., Decker, S., 2007. The Future of Social Networks on the Internet: The Need for Semantics. IEEE Internet Computing 11, 86–90.

Breslin, J.G., Passant, A., Decker, S., 2009. Motivation for Applying Semantic Web Technologies to the Social Web, in: The Social Semantic Web. Springer Berlin Heidelberg, pp. 11–20.

Broekstra, J., Kampman, A., 2003. SeRQL: A Second Generation RDF Query Language, in: Proceedings of the SWAD-Europe Workshop on Semantic Web Storage and Retrieval. Vrije Universiteit, Amsterdam, Netherlands.

Broekstra, J., Kampman, A., Harmelen, F. van, 2002. Sesame: An Architecture for Storing and Querying RDF and RDF Schema, in: Proceedings of the First International Semantic Web Conference, LNCS. Presented at the ISWC 2002, Springer-Verlag, pp. 54–68.

Būmans, G., Cerāns, K., 2010. RDB2OWL: a practical approach for transforming RDB data into RDF/OWL, in: Proceedings of the 6th International Conference on Semantic Systems, I-SEMANTICS '10. ACM, New York, NY, USA, pp. 25:1–25:3.

Busch, M., Koch, N., 2009. Rich Internet Applications State-of-the-Art. Ludwig-Maximilians-Universität München, Munich, Germany.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., 1996. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Wiley, Chichester, UK.

Cachero, C., Meliá, S., Genero, M., Poels, G., Calero, C., 2007. Towards improving the navigability of Web applications: a model-driven approach. European Journal of Information Systems 16, 420–447.

Casteleyn, S., Plessers, P., De Troyer, O., 2006. On generating content and structural annotated websites using conceptual modeling, in: Proceedings of the 25th International Conference on Conceptual Modeling, ER'06. Springer-Verlag, Berlin, Heidelberg, pp. 267–280.

Ceri, S., Brambilla, M., Fraternali, P., 2009. The History of WebML: Lessons Learned from 10 Years of Model-Driven Development of

Web Applications, in: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (Eds.), Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos. Springer-Verlag, Berlin, Heidelberg, pp. 273–292.

Ceri, S., Fraternali, P., Bongio, A., 2000. Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33, 137–157.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M., 2002. Designing Data-Intensive Web Applications. Elsevier, Amsterdam, Netherlands.

Corcho, O., López-Cima, A., Gómez-Pérez, A., 2006. A platform for the development of semantic web portals, in: Proceedings of the 6th International Conference on Web Engineering (ICWE'06). ACM, New York, NY, USA, pp. 145–152.

Cranefield, S., Purvis, M.K., 1999. UML as an Ontology Modelling Language, in: Proceedings of the IJCAI-99 Workshop on Intelligent Information Integration, Held on July 31, 1999 in Conjunction with the Sixteenth International Joint Conference on Artificial Intelligence City Conference Center, Stockholm, Sweden, CEUR Workshop Proceedings. Presented at the IJCAI-99 Workshop on Intelligent Information Integration, Stockholm, Sweden.

Choudhary, S., Dincturk, M.E., Mirtaheri, S.M., Moosavi, A., Von Bochmann, G., Jourdan, G.-V., Onut, I.-V., 2012. Crawling Rich Internet Applications: The State of the Art, in: Proceedings of the 22nd Annual International Conference Hosted by the Centre for Advanced Studies Research, IBM Canada Software Laboratory. Presented at the The 22nd annual international conference hosted by the Centre for Advanced Studies Research, IBM Canada Software Laboratory.

D' Aquin, M., Motta, E., Sabou, M., Angeletou, S., Gridinoc, L., Lopez, V., Guidi, D., 2008. Towards a new generation of semantic web applications. IEEE Intelligent Systems 23, 20–28.

Davenport, T.H., 2000. The Future of Enterprise System-Enabled Organizations. Information Systems Frontiers 2, 163–180.

De Souza Bomfim, M.H., Schwabe, D., 2011. Design and Implementation of Linked Data Applications Using SHDM and Synth, in: Web

Engineering - 11th International Conference, ICWE 2011, Paphos, Cyprus, June 20-24, 2011, Lecture Notes in Computer Science. Presented at the 11th International Conference (ICWE 2011), Springer, Paphos, Cyprus, pp. 121–136.

De Troyer, O., Casteleyn, S., Plessers, P., 2007. WSDM: Web Semantics Design Method, in: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (Eds.), Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series. Springer, London, pp. 303–351.

De Troyer, O., Leune, K., 1998. WSDM: a user centered design method for Web sites, in: Proceedings of the 7th International World Wide Web Conference. Presented at the Word Wide Web Conference (WWW'98), Elsevier, Brisbane, Australia, pp. 85–94.

Djuric, D., Gasevic, D., Devedzic, V., Damjanovic, V., 2004. UML Profile for OWL, in: Web Engineering - 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004, Proceedings, Lecture Notes in Computer Science. Presented at the 4th International Conference on Web Engineering (ICWE 2004), Springer, Munich, Germany, pp. 607–608.

Fialho, A.T.S., Schwabe, D., 2007. Enriching hypermedia application interfaces. Lecture Notes in Computer Science 4607, 188–193.

Fowler, M., 2002. Patterns of Enterprise Application Architecture. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Fowler, M., 2004. Presentation Model [WWW Document]. URL http://martinfowler.com/eaaDev/PresentationModel.html

Frasincar, F., Houben, G.-J., Barna, P., 2010. Hypermedia presentation generation in Hera. Inf. Syst. 35, 23–55.

Fraternali, P., Comai, S., Bozzon, A., Toffeti Carughi, G., 2010a. Engineering rich internet applications with a model-driven approach. ACM Transactions on the Web 4, 1–47.

Fraternali, P., Tisi, M., Silva, M., Frattini, L., 2010b. Building Community-based Web Applications with a Model-Driven Approach and Design Patterns. Politecnico di Milano. Available from:
http://webml.org/webml/upload/ent5/1/CommunityPatterns-final.pdf

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gasevic, D., Djuric, D., Devedzic, V., 2005. Bridging MDA and OWL Ontologies. J. Web Eng. 4, 118–143.

Gasevic, D., Djuric, D., Devedzic, V., 2007. MDA-based Automatic OWL Ontology Development. STTT 9, 103–117.

Gerber, A., Van der Merwe, A., Barnard, A., 2008. A Functional Semantic Web Architecture, in: Proceedings of the 5th European Semantic Web Conference. Presented at the ESWC'08, pp. 273–287.

Gómez, J., Cachero, C., Pastor, O., 2001. Conceptual Modeling of Device-Independent Web Applications. IEEE MultiMedia 8, 26–39.

Gómez-Pérez, A., Fernández-López, M., Corcho, O., 2007. Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. (Advanced Information and Knowledge Processing). Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Gossman, J., 2005. Introduction to Model/View/ViewModel pattern for building WPF apps. MSDN Blogs - Tales from the Smart Client.

Groza, T., Handschuh, S., Moeller, K., Grimnes, G., Sauermann, L., Minack, E., Mesnage, C., Jazayeri, M., Reif, G., Gudjonsdottir, R., 2007. The NEPOMUK Project - On the way to the Social Semantic Desktop, in: Pellegrini, T., Schaffert, S. (Eds.), Proceedings of the 7th International Conference on Semantic Systems (I-SEMANTICS' 07). Presented at the 7th International Conference on Semantic Systems, JUCS, pp. 201–211.

Gruber, T.R., 1995. Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies 5–6, 907–928.

Guarino, N., 1998. Formal Ontology in Information Systems, in: Proceedings of the International Conference on Formal Ontology in Information Systems. Presented at the FOIS'98, IOS Press, Trento, Italy, pp. 3–15.

Heath, T., Bizer, C., 2011. Linked Data. Evolving the Web into a Global Data Space, 1st ed, Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool.

Heitmann, B., Kinsella, S., Hayes, C., Decker, S., 2009. Implementing Semantic Web applications: reference architecture and challenges, in: Proceedings of the 5th International Workshop on Semantic Web-Enabled Software Engineering. Presented at the SWESE'09, Virginia, USA, pp. 16–30.

Hermida, J.M., Meliá, S., Martinez, J.-J., Montoyo, A., Gómez, J., 2012a. Developing Semantic Rich Internet Applications with the Sm4RIA Extension for OIDE, in: ICWE Workshops 2012, Lecture Notes in Computer Science. Presented at the Workshop on Model-Driven and Agile Engineering for the Web (MDWE 2012), Springer Berlin Heidelberg, Belin, Germany, pp. 20–25.

Hermida, J.M., Meliá, S., Montoyo, A., Gómez, J., 2011a. Developing Rich Internet Applications as Social Sites on the Semantic Web: A Model-Driven Approach. IJSSOE 2, 21–41.

Hermida, J.M., Meliá, S., Montoyo, A., Gómez, J., 2011b. Developing Semantic Rich Internet Applications Using a Model-Driven Approach, in: Web Information Systems Engineering - WISE 2010 Workshops - WISE 2010 International Symposium WISS, and International Workshops CISE, MBC, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, pp. 198–211.

Hermida, J.M., Meliá, S., Montoyo, A., Gómez, J., 2012b. Sm4RIA Extension for OIDE: Desarrollo de Rich Internet Applications en la Web Semántica., in: Actas De Las "XVII Jornadas De Ingeniería Del Software y Bases De Datos (JISBD)". Presented at the XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD'12), Universidad de Almería, Almería, Spain, pp. 123–126.

Hermida, J.M., Meliá, S., Montoyo, A., Gómez, J., 2013. Applying Model-Driven Engineering to the Development of Rich Internet Applications for Business Intelligence. ISF in-press.

Hermida, J.M., Montoyo, A., Gómez, J., 2009. Improving Semantic Web Applications with Navigational Semantics, in: Natural Language Processing and Information Systems, 14th International Conference on Applications of Natural Language to Information Systems, NLDB 2009, Saarbrücken, Germany, June 24-26, 2009. Revised Papers, Lecture Notes in Computer Science. Presented at the 14th International Conference on Applications of Natural Language to Information Systems, NLDB 2009, Springer, Saarbrücken, Germany, pp. 291–292.

Horrocks, I., Parsia, B., Patel-Schneider, P., Hendler, J., 2005. Semantic Web Architecture: Stack or Two Towers? Principles and Practice of Semantic Web Reasoning 37–41.

Houben, G.-J., Barna, P., Frasincar, F., Vdovjak, R., 2003. Hera: Development of Semantic Web Information Systems, in: Proceedings of the 3rd International Conference on Web Engineering. Presented at the ICWE 2003, Springer-Verlag, pp. 529–538.

Isakowitz, T., Bieber, M., Vitali, F., 1998. Web information systems. Commun. ACM 41, 78–80.

Jin, Y., Decker, S., Wiederhold, G., 2001. OntoWebber: Model-Driven Ontology-Based Web Site Management. Presented at the Semantic Web Working Symposium (SWWS), Stanford University, pp. 529–547.

Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M., 2006. Lifting metamodels to ontologies: a step to the semantic integration of modeling languages, in: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, MoDELS'06. Presented at the 9th international conference on Model Driven Engineering Languages and Systems, Springer-Verlag, Berlin, Heidelberg, pp. 528–542.

Kinsella, S., Passant, A., Breslin, J.G., Decker, S., Jaokar, A., 2009. The Future of Social Web Sites: Sharing Data and Trusted Applications with Semantics. Advances in Computers 76, 121–175.

Knapp, A., Koch, N., Wirsing, M., 2004. Modeling the Structure of Web Applications with ArgoUWE, in: Web Engineering, 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004. Proceedings, Lecture Notes in Computer Science. Presented at the Modeling the Structure of Web Applications with ArgoUWE, Springer Berlin Heidelberg, Munich, Germany, pp. 615–616.

Koch, N., Kraus, A., 2002. The expressive power of UML-based Web Engineering, in: Proceecings of the 2nd International Workshop on Web Oriented Software Technology (IWWOST'2002). Presented at the 2nd International Workshop on Web Oriented Software Technology (IWWOST'2002), pp. 105–119.

Koch, N., Kraus, A., 2003. Towards a Common Metamodell for the Development of Web Appliactions, in: Web Engineering, International Conference, ICWE 2003, Oviedo, Spain, July 14-18, 2003, Proceedings, Lecture Notes in Computer Science. Presented at the 3rd International Conference on Web Engineering, Springer-Verlag, Oviedo, Spain, pp. 497–506.

Koch, N., Pigerl, M., Zhang, G., Morozova, T., 2009. Patterns for the Model-Based Development of RIAs, in: Proceedings of the 9th International Conference on Web Engineering, Lecture Notes in Computer Science. Presented at the 9th International Conference on Web Engineering, Springer-Verlag, Berlin, Heidelberg, pp. 283–291.

Kozaki, K., Hayashi, Y., Sasajima, M., Tarumi, S., Mizoguchi, R., 2008. Understanding Semantic Web Applications. Lecture Notes in Computer Science 5367, 524–539.

Laurent, W., 2010. Flash And Silverlight.New Staples Of Rich Visual BI Applications. Dashboard Insight.

Lausen, H., Ding, Y., Stollberg, M., Fensel, D., Hernandez, R.L., Han, S.-K., 2005. Semantic web portals: state-of-the-art survey. Journal of Knowledge Management 9, 40–49.

Lima, F., Schwabe, D., 2003. Modeling applications for the semantic Web. Lecture Notes in Computer Science 2722.

Linaje, M., Lozano-Tello, A., Perez-Toledano, M.A., Preciado, J.C., Rodriguez-Echeverria, R., Sánchez-Figueroa, F., 2011. Providing RIA user interfaces with accessibility properties. Journal of Symbolic Computation 46, 207 – 217.

Linaje, M., Lozano-Tello, A., Preciado, J.C., Rodríguez, R., Sánchez-Figueroa, F., 2009a. Obtaining accessible RIA UIs by combining RUX-Method and SAW, in: Proceedings of the International Workshop on Automated Specification and Verification of Web Systems. pp. 85–97.

Linaje, M., Preciado, J.C., Morales-Chaparro, R., Rodríguez-Echeverría, R., Sánchez-Figueroa, F., 2009b. Automatic Generation of RIAs Using RUX-Tool and Webratio, in: Proceedings of the 9th International Conference on Web Engineering, Lecture Notes in Computer Science. Springer-Verlag, San Sebastián, Spain, pp. 501–504.

Linaje, M., Preciado, J.C., Sánchez-Figueroa, F., 2007. A method for model based design of rich internet application interactive user interfaces, in: Proceedings of the 7th International Conference on Web Engineering, Lecture Notes in Computer Science. Presented at the ICWE'07, Springer-Verlag, Como, Italy, pp. 226–241.

Linaje Trigueros, M., Preciado, J.C., Sánchez-Figueroa, F., 2007. Engineering Rich Internet Application User Interfaces over Legacy Web Models. IEEE Internet Computing 11, 53–59.

Machado, L., Filho, O., Ribeiro, J., 2009. UWE-R: an extension to a web engineering methodology for rich internet applications. WSEAS Trans. Info. Sci. and App. 6, 601–610.

Martinez, Y., Cachero, C., Meliá, S., 2013. MDD vs. traditional software development: A practitioner's subjective perspective. Information and Software Technology 55, 189–200.

Meliá, S., 2007. WebSA: Un método de desarrollo dirigido por modelos de arquitectura para aplicaciones Web (PhD Thesis). University of Alicante, Alicante. Spain.

Meliá, S., Gómez, J., Pérez, S., Díaz, O., 2008. A model-driven development for GWT-based rich Internet applications with OOH4RIA, in: Proceedings of the 2008 Eighth International Conference on Web Engineering (ICWE 2008). IEEE Computer Society, Yorktown Heights, New York, USA, pp. 13–23.

Meliá, S., Gómez, J., Pérez, S., Díaz, O., 2010a. Architectural and technological variability in Rich Internet Applications. IEEE Internet Computing 14, 24–32.

Meliá, S., Martinez, J.-J., Mira, S., Osuna, J.-A., Gómez, J., 2010b. An Eclipse Plug-in for Model-Driven Development of Rich Internet Applications. Lecture Notes in Computer Science 6189, 514–517.

Mesbah, A., Bozdag, E., Deursen, A. van, 2008. Crawling AJAX by Inferring User Interface State Changes, in: Proceedings of the Eighth International Conference on Web Engineering, ICWE 2008, 14-18 July 2008, Yorktown Heights, New York, USA. Presented at the 8th International Conference on Web Engineering (ICWE 2008), IEEE, New York, NY, USA, pp. 122–134.

Mesbah, A., Deursen, A. van, Lenselink, S., 2012. Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes. TWEB 6, 3.

Mesbah, A., Van Deursen, A., 2007. Migrating multi-page Web applications to single-page AJAX interfaces, in: Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR 2007). Amsterdam, Netherlands, pp. 181–190.

Mikroyannidis, A., 2007. Toward a Social Semantic Web. Computer 40, 113–115.

Murugesan, S., 2008. Web Application Development: Challenges and the Role of Web Engineering, in: Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series. Springer, London, pp. 7–32.

Object Management Group, 2003. MDA Guide Version 1.0.1. Available from: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

Object Management Group, 2008. Software Process Engineering Meta-Model, version 2.0. OMG. Available from: http://www.omg.org/cgi-bin/doc?formal/08-04-01.pdf

Object Modeling Group, 2009. Ontology Definition Metamodel Version 1.0. OMG. Available from: http://www.omg.org/spec/ODM/1.0/PDF

Penela, V., Álvaro, G., Ruiz, C., Córdoba, C., Carbone, F., Castagnone, M., Gómez-Pérez, J.M., Contreras, J., 2011. miKrow: semantic intra-enterprise micro-knowledge management system, in: Proceedings of the 8th Extended Semantic Web Conference on The Semanic Web: Research and Applications - Volume Part II, ESWC'11. Springer-Verlag, Berlin, Heidelberg, pp. 154–168.

Plessers, P., Casteleyn, S., Yesilada, Y., De Troyer, O., Stevens, R., Harper, S., Goble, C., 2005. Accessibility: a Web engineering approach, in: Proceedings of the 14th International Conference on World Wide Web, WWW '05. ACM, New York, NY, USA, pp. 353–362.

Preciado, J.C., Linaje, M., Comai, S., Sánchez-Figueroa, F., 2007. Designing rich internet applications with Web engineering methodologies, in: Proceedings of the Third International Workshop on Automated Specification and Verification of Web

Systems (WWV 2007). San Servolo island, Venice, Italy, p. 23?–30.

Preciado, J.C., Linaje, M., Morales-Chaparro, R., Sanchez-Figueroa, F., Zhang, G., Kroiss, C., Koch, N., 2008. Designing Rich Internet Applications Combining UWE and RUX-Method, in: Proceedings of the 2008 Eighth International Conference on Web Engineering. Presented at the ICWE ’08, IEEE Computer Society, New York, USA, pp. 148–154.

Raspal, N., 2010. Why RIA allows a New Breed of Business Intelligence Solution. Dashboard Insight.

Rovan, L., Jagust, T., Baranovic, M., 2011. Defining Categories and Functionalities of Semantic Web Applications. International Journal of Systems Applications, Engineering and Development 5, 245–253.

Schwabe, D., Rossi, G., 1998. An Object Oriented Approach to Web-Based Applications Design. TAPOS 4, 207–225.

Silva Parreiras, F., Staab, S., Winter, A., 2007. On marrying ontological and metamodeling technical spaces, in: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC-FSE ’07. Presented at the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM, New York, NY, USA, pp. 439–448.

Simperl, E., Thurlow, I., Warren, P., Dengler, F., Davies, J., Grobelnik, M., Mladenić, D., Gomez-Perez, J.M., Moreno, C.R., 2010. Overcoming Information Overload in the Enterprise: The Active Approach. IEEE Internet Computing 14, 39–46.

Smith, J., 2009. WPF Apps With The Model-View-ViewModel Design Pattern. MSDN Magazine 24.

Software & Information Industry Association, 2001. Software as a Service: Strategic Backgrounder. Available from: http://www.siia.net/estore/pubs/SSB-01.pdf

Toffetti Carughi, G., 2007. Conceptual Modeling and Code Generation of Data-Intensive Rich Internet Applications (PhD Thesis). Politecnico di Milano, Milan, Italy.

Toffetti, G., Comai, S., Preciado, J.C., Linaje, M., 2011. State-of-the Art and trends in the Systematic Development of Rich Internet Applications. J. Web Eng. 10, 70–86.

Trowbridge, D., Mancini, D., Quick, D., Hohpe, G., Newkirk, J., Lavigne, D., 2003. Enterprise Solution Patterns Using Microsoft .Net: Version 2.0®: Patterns & Practices. Microsoft Press, Redmond, WA, USA.

Urbieta, M., Rossi, G., Ginzburg, J., Schwabe, D., 2007. Designing the Interface of Rich Internet Applications, in: Fifth Latin American Web Congress (LA-Web 2007), 31 October - 2 November 2007, Santiago De Chile. Presented at the Fifth Latin American Web Congress (LA-Web 2007), IEEE Computer Society, Santiago de Chile, Chile, pp. 144–153.

Valverde, F., Pastor, O., 2008. Applying Interaction Patterns: Towards a Model-Driven Approach for Rich Internet Applications Development, in: Gaedke, M., Bieliková, M. (Eds.), Proceedings of the 7th International Workshop on Web-Oriented Software Technologies. Presented at the 7th International Workshop on Web-Oriented Software Technologies, New York, United States, pp. 13–18.

Valverde Giromé, F., 2010. OOWS 2.0: un método de ingeniería web dirigido por modelos para la producción de aplicaciones web 2.0 (PhD Thesis). Universidad Politécnica de Valencia, Valencia, Spain.

Van der Sluijs, K., Houben, G.-J., Broekstra, J., Casteleyn, S., 2006. Hera-S: Web design using sesame, in: Proceedings of the Sixth International Conference on Web Engineering (ICWE 2006). Palo Alto, California, USA, pp. 337–344.

Vdovjak, R., Frasincar, F., Houben, G.-J., Barna, P., 2003. Engineering Semantic Web Information Systems in Hera. Journal on Web Engineering 2, 3–26.

Volz, R., Handschuh, S., Staab, S., Stojanovic, L., Stojanovic, N., 2004. Unveiling the hidden bride: deep annotation for mapping and migrating legacy data to the Semantic Web. J. Web Sem. 1, 187–206.

W3C Model-based User Interfaces Incubator Group, 2009. SHDM - Semantic Hypermedia Design Method. W3C. Available from:

http://www.w3.org/2005/Incubator/model-based-ui/wiki/SHDM_-_Semantic_ Hypermedia_Design_Method

White, C., 2009. Using Rich Internet Applications in Business Intelligence. Technology Transfer.

World Wide Web Consortium, 2004. OWL Web Ontology Language Overview. W3C Recommendation. W3C. Available from: http://www.w3.org/TR/owl-features/

World Wide Web Consortium, 2008a. RDFa in XHTML: Syntax and Processing. W3C Recommendation. W3C. Available from: http://www.w3.org/TR/rdfa-syntax/

World Wide Web Consortium, 2008b. SPARQL protocol for RDF, W3C Recommendation. W3C. Available from: http://www.w3.org/TR/rdf-sparql-protocol/

World Wide Web Consortium, 2011. Accessible Rich Internet Applications (WAI-ARIA) 1.0, W3C Candidate Recommendation. W3C. Available from: http://www.w3.org/TR/wai-aria/

Xu, Z., Zhang, S., Dong, Y., 2006. Mapping between Relational Database Schema and OWL Ontology for Deep Annotation, in: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, WI '06. IEEE Computer Society, Washington, DC, USA, pp. 548–552.

Annex A. Scientific Contributions

This annex lists the contributions to scientific journals and conferences of the author during the period of development of this thesis.

Articles in international journals:

Hermida, J.M.; Meliá, S.; Montoyo, A.; Gómez, J. **Applying Model-Driven Engineering to the Development of Rich Internet Applications for Business Intelligence.** Information System Frontiers. DOI 10.1007/s10796-012-9402-9. 2013.

Balahur, A.; Hermida, J.M.; Montoyo, A. **Detecting Implicit Expressions of Emotion in Text: A Comparative Analysis.** Decision Support Systems 53(4), 742-753, 2012.

Balahur, A.; Hermida, J.M.; Montoyo, A. **Building and Exploiting EmotiNet, a Knowledge Base for Emotion Detection Based on the Appraisal Theory Model.** IEEE Transactions on Affective Computing 3(1), 88- 101, 2012.

Hermida, J.M.; Meliá, S.; Montoyo, A.; Gómez, J. **Developing Rich Internet Applications as Social Sites on the Semantic Web: A Model- Driven Approach.** International Journal of Systems and Service-Oriented Engineering (IJSSOE) 2(4), 21-41, 2011.

Contributions to international conferences and workshops:

Hermida, J.M.; Meliá, S.; Martínez, J.; Montoyo, A.; Gómez, J. **Developing Semantic Rich Internet Applications with the S^{m}4RIA Extension for OIDE.** In proceedings of the 8th workshop on Model-Driven and Agile Engineering for the Web (MDWE 2012) held in conjunction with the 12th International Conference on Web Engineering (ICWE 2012). Lecture Notes in Computer Science 7703, 20–25. 2012. ISSN 0302-9743

Balahur, A.; Hermida, J.M.; Montoyo, A. **Detecting Implicit Expressions of Sentiment in Text Based on Commonsense Knowledge.** In proceedings of the 2nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis (WASSA 2011) held with ACL 2011. Pages 53-60. 2011. `http://aclweb.org/anthology-new/W/W11/W11-1704.pdf`

Balahur, A.; Hermida, J.M.; Montoyo, A. **Detecting Emotions in Social Affective Situations Using the EmotiNet Knowledge Base.** In proceedings of the 8th International Symposium on Neural Networks (ISNN 2011). Lecture Notes in Computer Science, 6677, 611-620, 2011.

Balahur, A.; Hermida, J.M.; Montoyo, A.; Muñoz, R. **EmotiNet: A Knowledge Base for Emotion Detection in Text Built on the Appraisal Theory.** In proceedings of the 16th International Conference on Applications of Natural Language to Information Systems (NDLB 2011). Lecture Notes in Computer Science 6716, 27-39, 2011.

Romá-Ferri, M.T.; Hermida, J.M.; Palomar, M. **OntoFIS as a NLP Resource in the Drug-Therapy Domain: Design Issues and Solutions Applied.** In proceedings of the 16th International Conference on Applications of Natural Language to Information Systems (NDLB 2011). Lecture Notes in Computer Science 6716, 125-136, 2011.

Hermida, J.M.; Meliá, S.; Montoyo, A.; Gómez, J. **Rich Internet Applications on the Semantic Web: A Model-Driven Approach.** Poster at the 7th Extended Semantic Web Conference (ESWC 2011). 2011

Hermida, J.M.; Meliá, S.; Montoyo, A.; Gómez, J. **Developing Semantic Rich Internet Applications Using a Model-Driven Approach.** In proceedings of the 1st International Symposium on Web Intelligent Systems & Services (WISS 2010). Lecture Notes in Computer Science 6724, 198-211. 2011

Medina, D.; Hermida, J.M.; Montoyo, A.; Torres, A. **Assessing the quality of scientific publications as educational content on digital libraries.** In proceeedings of the 5th Iberian Conference on Information Systems and Technologies (CISTI 2010). Pages 390-395. 2010. ISBN: 978-1-4244-7227-7

Balahur, A.; Hermida, J.M.; Montoyo, A. **Information Need versus Privacy: Living in an Open Society and Its Consequences.** In proceedings of "A Global Surveillance Society?" 4th Biannual Surveillance and Society/SSN conference, London, UK, 2010. http://www.surveillancecultures.org/abstracts.html

Hermida, J. M.; Romá-Ferri, M. T.; Montoyo, A.; Palomar, M. **Reusing UML Class Models to Generate OWL Ontologies**. In proceedings of the 1st International Conference on Knowledge Engineering and Ontology Development (KEOD'09). INSTICC Press. Pages 281-286. 2009. ISBN 978-989-674-012-2.

Medina, D; Hermida, J.M.; Montoyo, A. **LOQEVAL: Propuesta de Evaluación de la calidad de los objetos de aprendizajes mediante ontologías**. In proceedings of the 6º Simposio Iberoamericano en Educación, Cibernética e Informática (SIECI 2009). Pages 337-340. 2009. ISBN 1-934272-66-3.

Medina, D.; Hermida Carbonell, J.; Montoyo, A. **A New Proposal for Learning Object Quality Evaluation Based on Ontology.** In proceedings of the International Conference on Education and New Learning Technologies (Edulearn'09). 2009. ISBN 978-84-612-9801-3.

Hermida, J.M.; Montoyo, A.; Gómez, J. **Improving Semantic Web Applications with Navigational Semantics.** In proceedings of the 14th International Conference on Application of Natural Language to Information Systems (NLDB'09). Lecture Notes in Computer Science, 5723, 291-292. 2009. `http://www.springerlink.com/content/48165x478u083908/`

Medina, D.; Hermida, J.M.; Montoyo, A. Towards a New Ontology-Driven Approach for the Extraction of Parameters in Learning Object Quality Evaluation Tasks. In Proceedings of the 4a Conferencia Ibérica de Sistemas e Tecnologias de Informaçao (CISTI 2009). Pages 509 – 512. 2009. ISBN 978-989-96247-0-2.

Contributions to national conferences:

Hermida, J.M.; Meliá, S.; Montoyo, A.; Gómez, J. **S^{m}4RIA Extension for OIDE: Desarrollo de Rich Internet Applications en la Web Semántica**. In proceedings of the XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2012), 123-126. Universidad de Almería, ISBN 978-84-15487-28-9. 2012. ***Best demo award***.

Hermida, J.M.; Meliá, S.; Montoyo, A.; Gómez, J. **Mejora de la interoperabilidad de las RIAs desde una perspectiva semántica: Un caso de estudio en la Web Social**. In proceedings of the XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2011). Pages 183–196. 2011

Romá-Ferri, M. T.; Hermida, J. M.; Montoyo, A.; Palomar, M. **Representación del Conocimiento Farmacoterapéutico: Diseño de una Ontología**. In proceedings of the XI Congreso Nacional de Informática de la Salud. Pages 240–247. 2008. ISBN 978-84-691-2468-0.

Annex B. MAIN ELEMENTS OF THE NAVIGATIONAL & VISUALISATION ONTOLOGIES

This annex describes the two fixed ontologies involved in the annotation model proposed for SRIA: the navigational ontology for representing the navigational aspects of a Web application and the visualisation ontology for representing the visual elements. As a remaining task, these ontologies should be aligned to existing ontologies in the Linked Data cloud (e.g., Dublin Core) in order to facilitate the interoperability among data sets and thus the reuse of the instances of both ontologies.

B.1. NAVIGATIONAL ONTOLOGY: NAVONTOLOGY

The NavOntology ontology defines the elements required to specify the components that play a role when a user surfs a Web site. The ontology was built using OWL 1.1 and published on the following Web site: `http://artemisa.dlsi.ua.es/ontology/sria/navOntology.owl`

The concepts of the main hierarchy of this ontology are the following (abstract classes are represented in *italics*):

- WebContainer
- *WebDocument*
    - MultimediaDocument
        - AudioDocument
        - Image
        - VideoDocument
        - Screenshot
    - PlainTextDocument
        - XmlDocument
        - WebPage
            - EntryPoint
            - AnnotatedWebPage
        - StyleSheet
- Parameter
    - ClientParameter

    - ServerParameter
- *WebLink*
    - StaticLink
        - Frame
    - NavigationalLink
        - AutomaticNavigationalLink
    - NavigationalPath

The subsequent paragraphs describe the most relevant elements of the ontology and its properties.

WebContainer

Name: Web container

Description: Represents a collection of Web documents.

Datatype properties:

WC_Type	
Description	Type of container
Domain	WebContainer
Range	String
WC_BaseURI	
Description	Base URI of the Web container
Domain	WebContainer
Range	String

Object properties:

WC_Documents	
Description	Documents contained in the Web container.
Domain	WebContainer
Range	Web Document

WebDocument

Name: Web document

Description: Represents a document that can be accessed from the Internet.

Object properties:

WD_Parameters	
Description	Parameters required to accessing the information of the document or part of it.
Domain	WebDocument
Range	Parameter

MultimediaDocument

Name: Multimedia document

Description: Represents any document with multimedia contents, e.g., audio, video or images.

Datatype properties:

MD_Description	
Description	Description of the multimedia contents.
Domain	MultimediaDocument
Range	String
MD_Format	
Description	Format of the contents. Depeding on the specific type of multimedia documents, designers could use a different collection of formats (e.g., png, jpeg for images and mpeg or mkv for videos.)
Domain	MultimediaDocument
Range	String
MD_Document	
Description	Title of the multimedia document.
Domain	MultimediaDocument
Range	String

AudioDocument

Name: Audio document

Description: Represents any document of audio in any format.

Datatype properties:

AD_Length	
Description	Length of the document (in seconds).
Domain	AudioDocument
Range	Integer

VideoDocument

Name: Video document

Description: Represents any document of video in any format.

Datatype properties:

VD_Length	
Description	Length of the document (in seconds).
Domain	VideoDocument

Range	Integer

Image

Name: Image

Description: Represents an image in any format.

Datatype properties:

I_Resolution	
Description	Resolution of the image in pixel per inch.
Domain	Image
Range	Integer

Screenshot

Name: Screen shot

Description: State of a certain Web page in a specific moment. A Web page can contain different screen shots given a single URL.

Object properties:

SS_WebPage	
Description	Web page to which it is related.
Domain	Screenshot
Range	WebPage

WebPage

Name: Web page

Description: Represent the concept of Web page from the user's perspective, defined as the HTML document visualised by the Web browser given a URL.

EntryPoint

Name: Entry point

Description: Initial Web page of a Web site.

AnnotatedWebPage

Name: Annotated Web page

Description: HTML Web page with annotations.

StyleSheet

Name: Style sheet

Description: Document that describes the visualisation of the elements contained in a Web page.

Parameter

Name: Parameter.

Description: Information required for performing some task.

Data type properties:

ParamName	
Description	Name of the parameter.
Domain	Parameter
Range	String
ParamType	
Description	Type of the parameter. Must be a XML type.
Domain	Parameter
Range	String
ParamValue	
Description	Value of the parameter.
Domain	Parameter
Range	String

WebLink

Name: Web link.

Description: Represents a generic link between two or more web documents.

NavigationalLink

Name: Navigational link

Description: Represents a link between two Web pages or two screenshots of the same web page. The activation of this link implies a change in the information visualised.

Object properties:

LinkInit	
Description	Initial node of the link.
Domain	NavigationalLink
Range	WebPage
LinkEnd	
Description	Ending node of the link.
Domain	NavigationalLink

Range	WebPage

AutomaticNavigationalLink

Name: Automatic navigational link

Description: Navigational link that does not require the interaction of the user for its activation.

NavigationalPath

Name: Navigational path

Description: Collection of navigational links, in which the ending of a link is the beginning of another.

Object properties:

Link	
Description	Current link in the path.
Domain	NavigationalPath
Range	NavigationalLink
NP_Last	
Description	Last step in the navigational path.
Domain	NavigationalPath
Range	NavigationalLink

B.2. VISUALISATION ONTOLOGY

The Visualisation ontology represents the elements visualised by users on the Web browser, i.e., it represents the elements of the user interface from the user's viewpoint.The ontology was built using OWL 1.0 and published on the following Web site: `http://artemisa.dlsi.ua.es/ontology/sria/visuOntology.owl`. This ontology was the basis of the Visualisation Ontology metamodel.

The concepts of the main hierarchy are the following (abstract classes are represented in *italics*):

1. *VisualElement*
    a. *BehaviouralElement*
        i. Action
        ii. Event
    b. *StructuralElement*
        i. Annotation

- ii. Property
- iii. Component
    1. Screenshot
    2. SimpleElement
        - a. ElementContainer
            - i. Panel
        - b. Button
        - c. CheckBox
        - d. TextBox
        - e. Chart
        - f. Label
        - g. HyperLink
        - h. Map
        - i. Combobox

The following paragraphs describe each of the concepts with their datatype and object properties.

VisualElement

Name: Visual element

Description: Any element involved in the visualisation of the user interface of a Web site.

Datatype properties:

VE_Name	
Description	Name given to the visual element
Domain	VisualElement
Range	String

BehaviouralElement

Name: Behavioural element

Description: Any element (visible or not) related to the behaviour of the user interface of the Web site.

Parent: VisualElement

Action

Name: Action

Description: Action performed by the user interface.

Parent: BehaviouralElement

Event

Name: Event

Description: Event triggered by the user on the interface.

Parent: BehaviouralElement

Run

Name: Run

Description: Invocation of a certain action after an event is triggered.

Parent: BehaviouralElement

Object properties:

Run_Event	
Description	Event triggered.
Domain	Run
Range	Event
Run_Action	
Description	Actions run after the event was triggered.
Domain	Run
Range	Action

StructuralElement

Name: Structural element

Description: Any element related to the visual structure of the Web site on which the information is embedded.

Parent: VisualElement

Annotation

Name: Annotation

Description: Textual annotation defined over a component of the user interface

Parent: StructuralElement

Datatype properties:

Ann_TargetUri	
Description	Web page to which it is related.
Domain	Screenshot
Range	WebPage

Property

Name: Property

Description: Property of a structural element.

Datatype properties:

P_Type	
Description	Type of the property (must be an XML type).
Domain	Property
Range	String
P_Value	
Description	Value of the property.
Domain	Screenshot
Range	WebPage

Component

Name: Component.

Description: Region of the user interface with a specific purpose.

Parent: StructuralElement

Object properties:

C_AvailableEvent	
Description	Events available for the component.
Domain	Component
Range	Event

ScreenShot

Name: Screen shot

Description: In rich interfaces, a screen shot defines a certain state of the user interface, characterised by the state of all the structural elements it contains.

Parent: Component

Object properties:

SS_Visualises	
Description	Elements visualised in a certain screen shot.
Domain	ScreenShot
Range	Component

SimpleElement

Name: Simple element

Description: Structural element of the user interface with a specific purpose. It has a collection of properties associated that define different features such as position, size, aesthetic features, as well as the events that can trigger.

Parent: Component

ElementContainer

Name: Element container

Description: Element that can contain other elements.

Parent: SimpleElement

EC_Contains	
Description	Elements contained by the current ElementContainer instance.
Domain	Screenshot
Range	SimpleElement

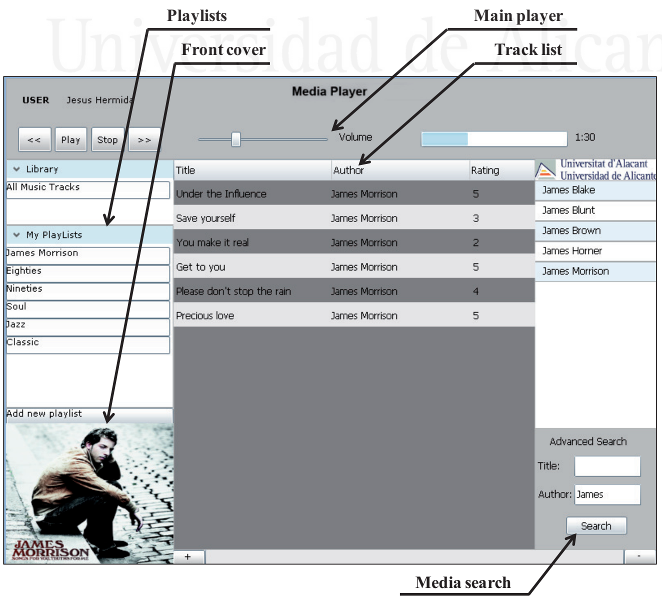
Annex C. Description of the SRIA Case Studies

This annex describes in detail the case studies not introduced in the main chapters of the manuscript. For the rest, it only includes a reference to the section containing the description.

C.1. Media player

The first case study addresses the development of an on-line media player using a SRIA as a platform (Hermida et al., 2011b). The SRIA media player will be able to play different types of media (mainly audio files) stored either locally or remotely on the Web. In addition, the application will be able to retrieve information related to the media played in each moment, or even search new media elements, through the Jamendo SPARQL service, which stores information about music elements as MusicOntology instances. A screenshot of the final application is shown in Figure C.1.

This application can be fount at `http://suma2.dlsi.ua.es/ooh4ria/sm4ria.html#uc`



Title	Author	Rating
Under the Influence	James Morrison	5
Save yourself	James Morrison	3
You make it real	James Morrison	2
Get to you	James Morrison	5
Please don't stop the rain	James Morrison	4
Precious love	James Morrison	5

James Blake
James Blunt
James Brown
James Horner
James Morrison

Advanced Search
Title:
Author: James
Search

Figure C.1. Screenshot of the user interface of the media player.

The user interface has been designed as simple as possible with regard to the number of elements and their visual appearance. This case study is focused on the new elements and processes of SRIA and, therefore, UI visual composition and effects are not relevant.

Before accessing the main functionalities of the application, users must have created a personal account. Despite the fact that this is not a key feature, the registration process is compulsory since the system might be closed to some types of users.

Once registered, they can access the main interface of the system through a authentication form. After this process, a list of the user's playlists or recently opened media elements is shown in the central area of the player. If the user opens a play list, its elements will appear within the same container.

As mentioned before, the application can only play audio tracks, which can be stored locally (in a limited storage provided by the SRIA) or externally (in other Web sites, e.g., Jamendo). Users can group their tracks into playlists using the UI options for adding or deleting elements to elements. Media elements can be simply played by clicking twice on the name of each element. Users can change the information to all his/her elements (personal playlists, music and video tracks) and can rate it according to his/her personal preferences.

The application UI contains three side menus: The first one, located on the left part, will show the library of albums, the user playlists and the front cover of the album or song selected in each moment. More specifically, in this menu, the interface will manage the user playlists and the options needed to create, save and delete them.

The main controllers (play/open, stop, fast forward, fast backward, volume control) are located on the top of the UI. The application also includes a progress bar showing the position within the track (minute and second) and the total length of the element played at a certain moment.

The last area of the application will include a search form, where users will be able to search and import into their playlist songs from external music sources, in this case, limited to Jamendo.

The application allows users to specify whether their personal data (user profile or music preferences) can be shared as ontology instances that might be reused by other systems. Security issues are out of the scope of this case study.

C.2. Social Network Site

This case study was introduced in Section 3.4.1 (page 54).

C.3. Social Network Site for Business Knowledge Management

This case study was introduced in Section 3.5.2 (page 60).

Annex D. Design Models Resulting From the Case Studies

This annex presents the S^m4RIA models obtained from the process of development of the two case studies not described in the chapters of this manuscript: the media player and the social network for the management of the working activity of an enterprise. The models are included in the same order as they were modelled, following the S^m4RIA process (or S^m4RIA-B).

D.1. Media Player

The following models can be used to develop the Media player case study with S^m4RIA, described in Annex C. Originally, they were introduced by Hermida et al. (Hermida et al., 2011b)

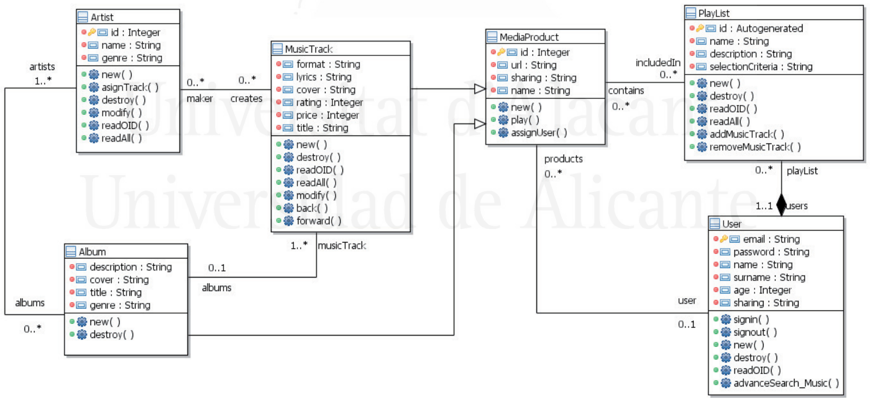
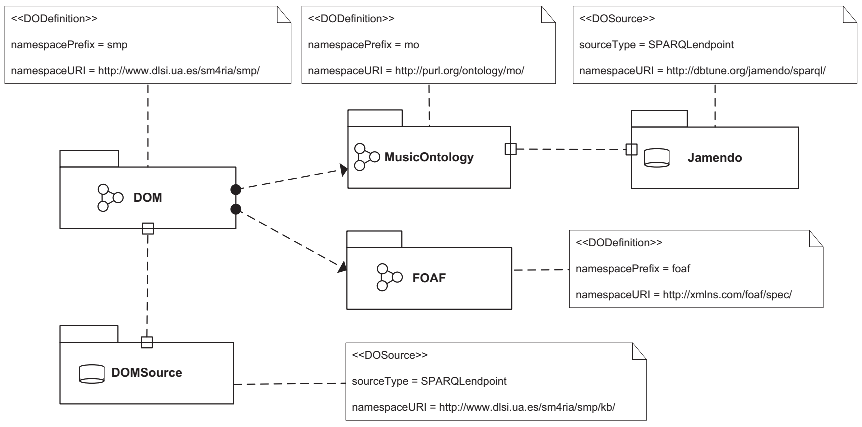


Figure D.2. Domain Model of the Media Player case study.



<<DODefinition>>
namespacePrefix = smp
namespaceURI = http://www.dlsi.ua.es/sm4ria/smp/
<<DODefinition>>
namespacePrefix = mo
namespaceURI = http://purl.org/ontology/mo/
<<DOSource>>
sourceType = SPARQLendpoint
namespaceURI = http://dbtune.org/jamendo/sparql/
MusicOntology
Jamendo
DOM
FOAF
<<DODefinition>>
namespacePrefix = foaf
namespaceURI = http://xmlns.com/foaf/spec/
DOMSource
<<DOSource>>
sourceType = SPARQLendpoint
namespaceURI = http://www.dlsi.ua.es/sm4ria/smp/kb/

Figure D.3. Ontology view of the Extended Domain Model of the Media Player case study.

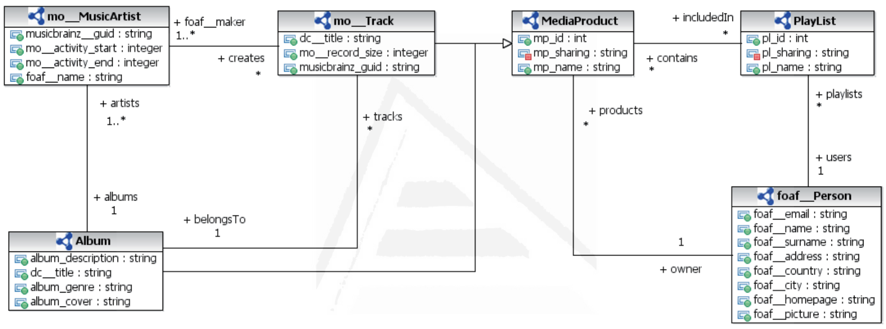


Figure D.4. Concept view of the Extended Domain Model for the Media Player case study.

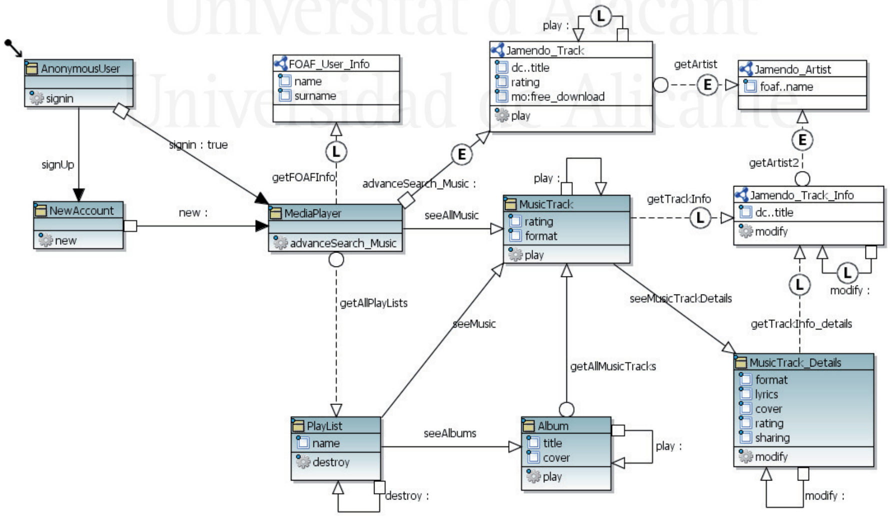
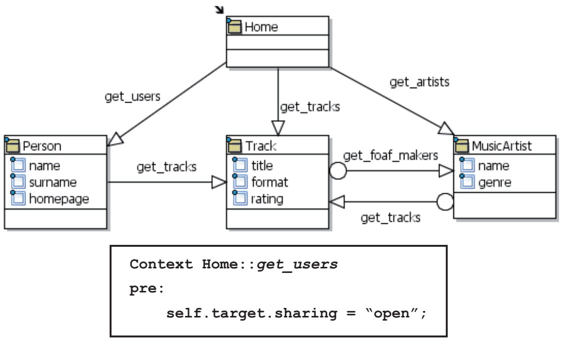


Figure D.5. View of the Extended Navigational Model for human users.



```
Context Home::get_users
pre:
    self.target.sharing = "open";
```

Figure D.6. View of the Extended Navigational Model for software agents.

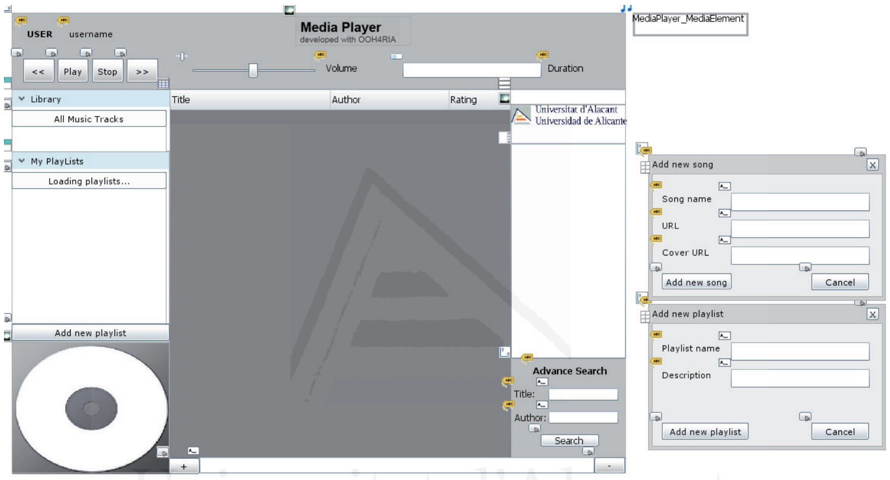


Figure D.7. Extended Presentation Model of the Media Player case study.

D.2. A SOCIAL APPLICATION FOR MANAGING BUSINESS KNOWLEDGE AS A RI@BI

The following models can be used to develop a Semantic Rich Internet Application in the Business Intelligence field. Specifically, they specify the design of a Social Network Site for Business Intelligence as described in Section 3.5.2 (page 60). They were firstly introduced by Hermida et al. (Hermida et al., 2013).

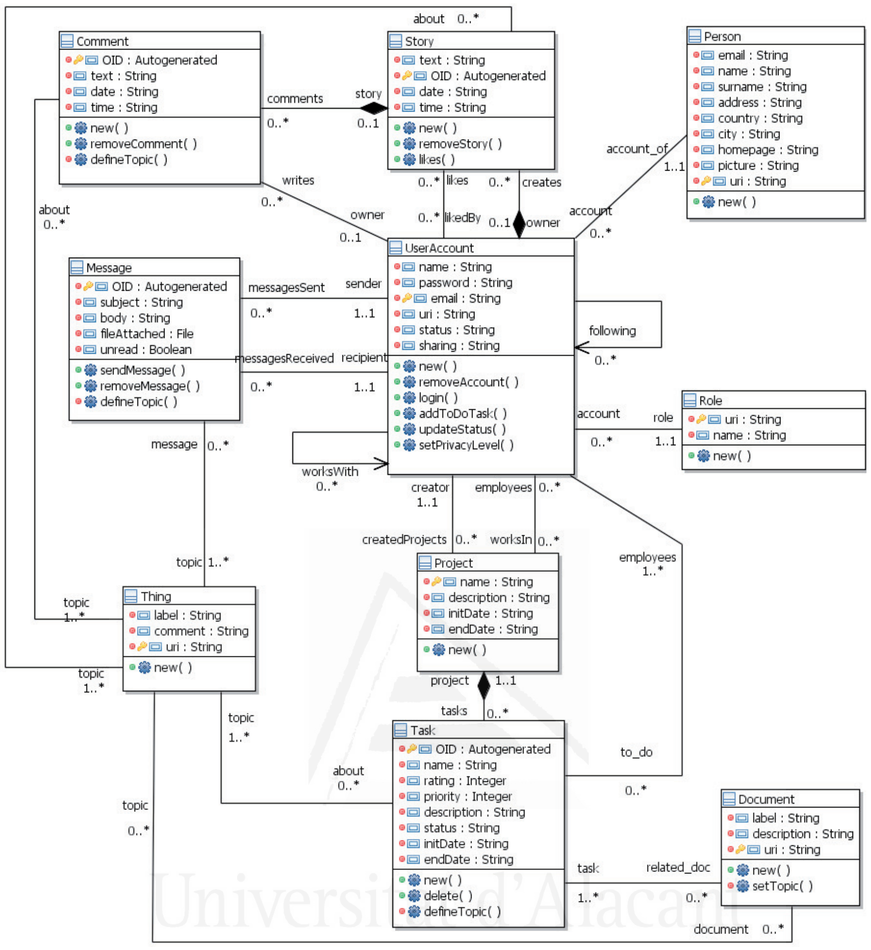


Figure D.8. Domain Model of the RI@BI case study.

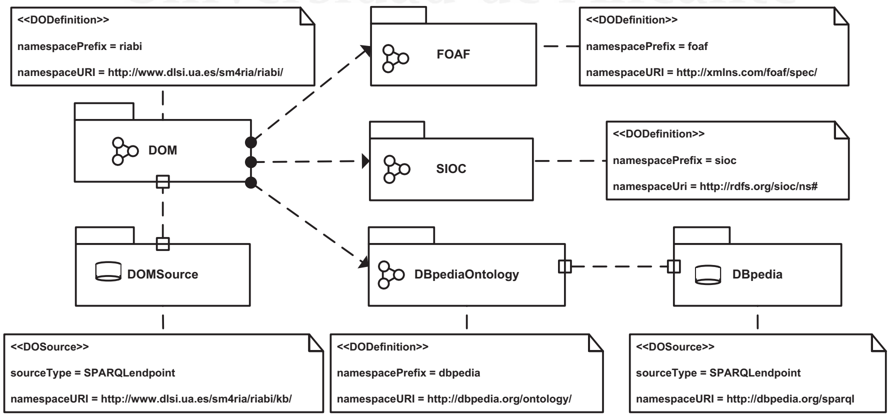

<<DODefinition>>
namespacePrefix = riabi
namespaceURI = http://www.dlsi.ua.es/sm4ria/riabi/
FOAF
<<DODefinition>>
namespacePrefix = foaf
namespaceURI = http://xmlns.com/foaf/spec/
DOM
SIOC
<<DODefinition>>
namespacePrefix = sioc
namespaceUri = http://rdfs.org/sioc/ns#
DOMSource
DBpediaOntology
DBpedia
<<DOSource>>
sourceType = SPARQLendpoint
namespaceURI = http://www.dlsi.ua.es/sm4ria/riabi/kb/
<<DODefinition>>
namespacePrefix = dbpedia
namespaceURI = http://dbpedia.org/ontology/
<<DOSource>>
sourceType = SPARQLendpoint
namespaceURI = http://dbpedia.org/sparql

Figure D.9. Ontology view of the Extended Domain Model for the RI@BI case study.

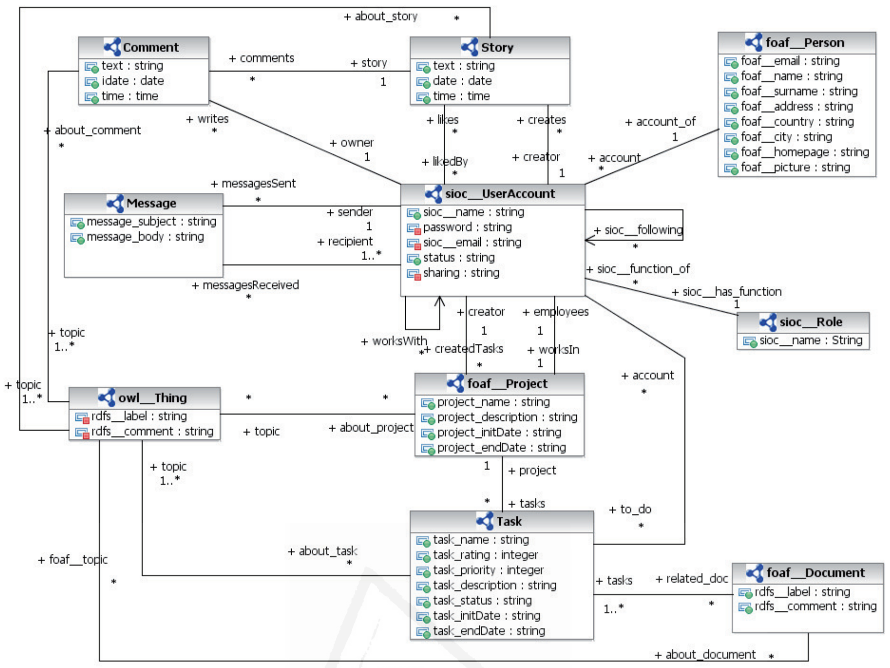


Figure D.10. Concept view of the Extended Domain Model for the RI@BI case study.

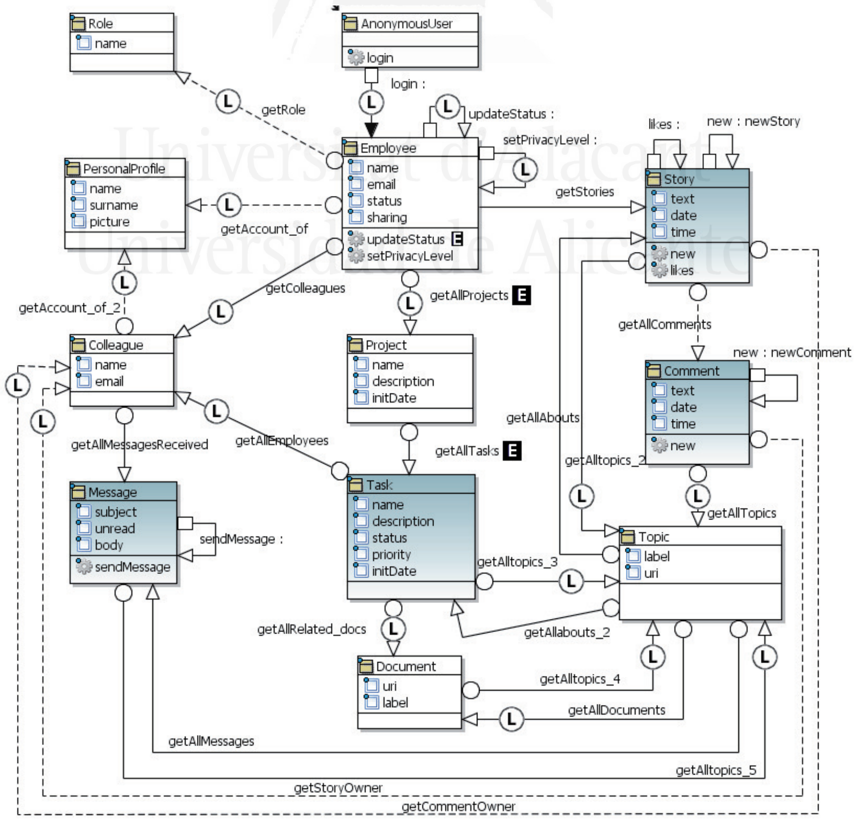


Figure D.11. Extended Navigational Model for the RI@BI case study.

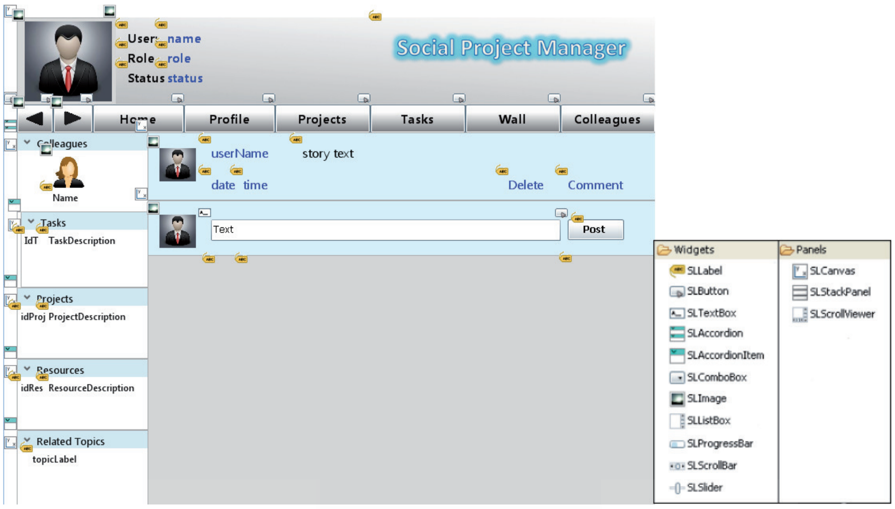


Figure D.12. Extended Presentation model showing the main screenshot for the RI@BI case study.

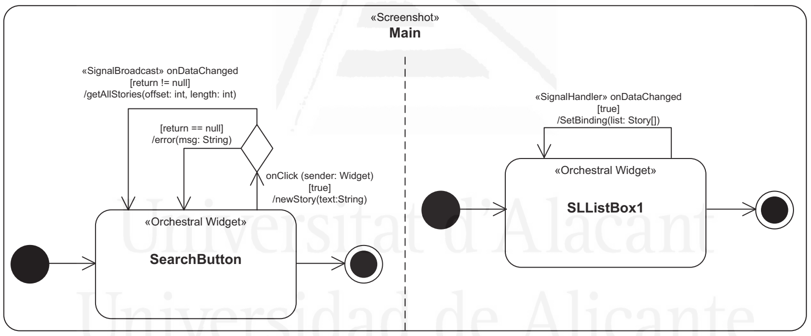


Figure D.13. Event-Condition-Action rule associated to the "Post" button of the user interface (part of the Extended Orchestration Model) of the RI@BI use case.

Annex E. The Extended Presentation Metamodel: Abstract Syntax

This annex introduces the complete abstract syntax of the Extended Presentation Metamodel as class diagrams. Given the size of the diagram, the diagram is split in three figures. The first figure (Figure E.14) shows an overview of the concrete syntax with its main elements. Figure E.15 and Figure E.16 illustrate the metaclass Widget and all its possible subclasses and properties.

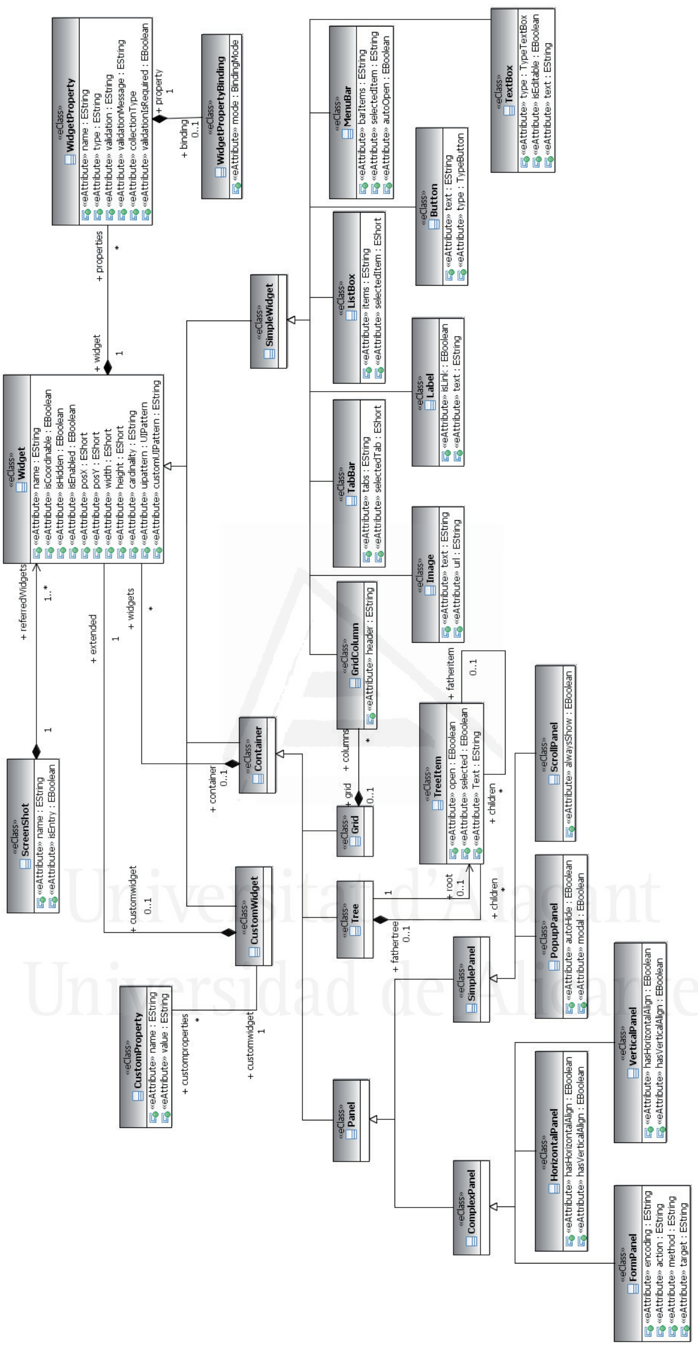

«eClass» WidgetProperty
«eAttribute» name : EString
«eAttribute» type : EString
«eAttribute» validation : EString
«eAttribute» validationMessage : EString
«eAttribute» collectionType
«eAttribute» validationIsRequired : EBoolean
«eClass» WidgetPropertyBinding
«eAttribute» mode : BindingMode
+ property 1
+ binding 0..1
+ properties *
+ widget 1
«eClass» Widget
«eAttribute» name : EString
«eAttribute» isCoordinable : EBoolean
«eAttribute» isHidden : EBoolean
«eAttribute» isEnabled : EBoolean
«eAttribute» posX : EShort
«eAttribute» posY : EShort
«eAttribute» width : EShort
«eAttribute» height : EShort
«eAttribute» cardinality : EString
«eAttribute» uipattern : UIPattern
«eAttribute» customUIPattern : EString
«eClass» SimpleWidget
«eClass» MenuBar
«eAttribute» barItems : EString
«eAttribute» selectedItem : EString
«eAttribute» autoOpen : EBoolean
«eClass» TextBox
«eAttribute» type : TypeTextBox
«eAttribute» isEditable : EBoolean
«eAttribute» text : EString
«eClass» Button
«eAttribute» text : EString
«eAttribute» type : TypeButton
«eClass» ListBox
«eAttribute» items : EString
«eAttribute» selectedItem : EShort
«eClass» Label
«eAttribute» isLink : EBoolean
«eAttribute» text : EString
«eClass» TabBar
«eAttribute» tabs : EString
«eAttribute» selectedTab : EShort
«eClass» Image
«eAttribute» text : EString
«eAttribute» url : EString
«eClass» GridColumn
«eAttribute» header : EString
+ referredWidgets 1..*
1
+ extended 1
+ widgets *
«eClass» ScreenShot
«eAttribute» name : EString
«eAttribute» isEntry : EBoolean
«eClass» Container
+ container 0..1
+ columns *
+ grid 0..1
«eClass» Grid
«eClass» TreeItem
«eAttribute» open : EBoolean
«eAttribute» selected : EBoolean
«eAttribute» Text : EString
+ fatheritem 0..1
+ children *
«eClass» ScrollPanel
«eAttribute» alwaysShow : EBoolean
+ customwidget 0..1
«eClass» CustomWidget
«eClass» Tree
+ root 0..1
1
+ children *
+ fathertree 0..1
«eClass» SimplePanel
«eClass» PopupPanel
«eAttribute» autoHide : EBoolean
«eAttribute» modal : EBoolean
«eClass» CustomProperty
«eAttribute» name : EString
«eAttribute» value : EString
+ customproperties *
+ customwidget 1
«eClass» Panel
«eClass» VerticalPanel
«eAttribute» hasHorizontalAlign : EBoolean
«eAttribute» hasVerticalAlign : EBoolean
«eClass» HorizontalPanel
«eAttribute» hasHorizontalAlign : EBoolean
«eAttribute» hasVerticalAlign : EBoolean
«eClass» ComplexPanel
«eClass» FormPanel
«eAttribute» encoding : EString
«eAttribute» action : EString
«eAttribute» method : EString
«eAttribute» target : EString

Figure E.14. Abstract components of the EMOF Extended Presentation Model.

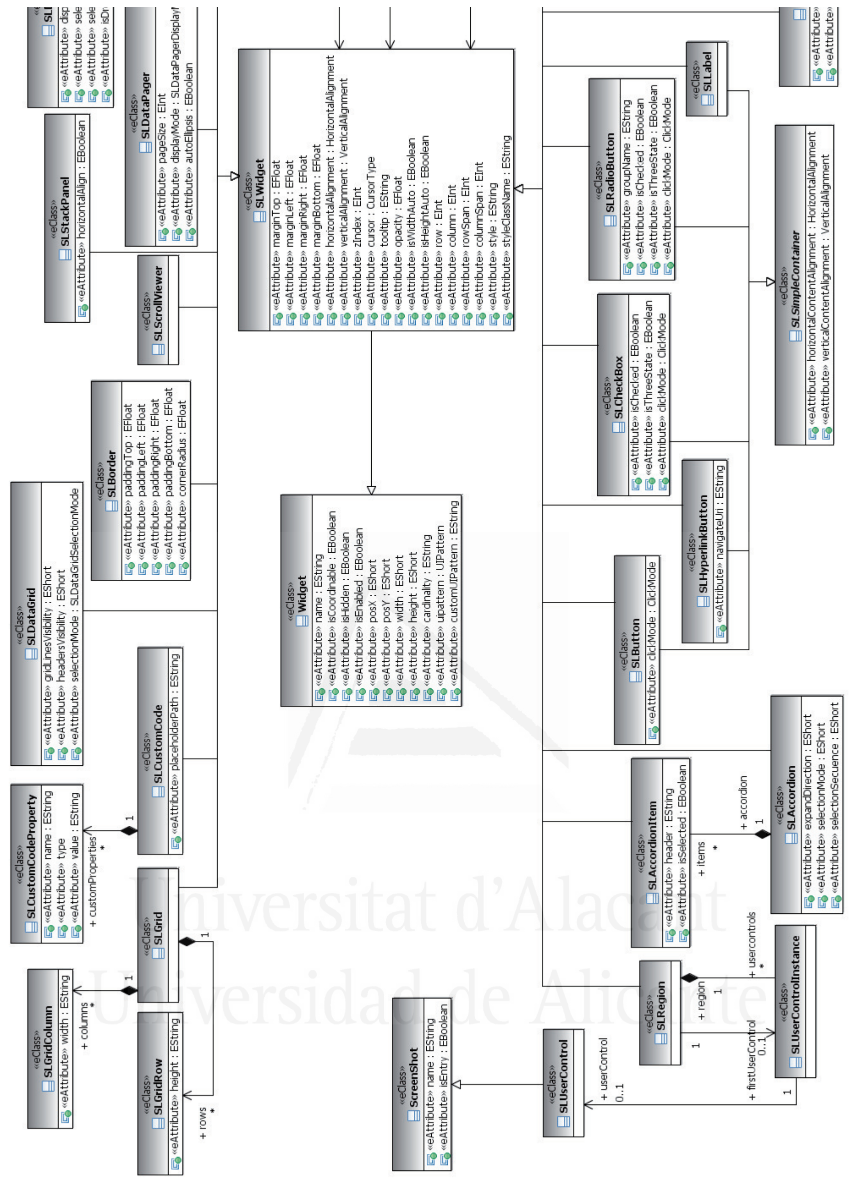

«eClass» SLStackPanel
«eAttribute» horizontalAlign : EBoolean
«eClass» SLDataPager
«eAttribute» pageSize : EInt
«eAttribute» displayMode : SLDataPagerDisplayM
«eAttribute» autoEllipsis : EBoolean
«eClass» SLScrollViewer
«eClass» SLWidget
«eAttribute» marginTop : EFloat
«eAttribute» marginLeft : EFloat
«eAttribute» marginRight : EFloat
«eAttribute» marginBottom : EFloat
«eAttribute» horizontalAlignment : HorizontalAlignment
«eAttribute» verticalAlignment : VerticalAlignment
«eAttribute» zIndex : EInt
«eAttribute» cursor : CursorType
«eAttribute» tooltip : EString
«eAttribute» opacity : EFloat
«eAttribute» isWidthAuto : EBoolean
«eAttribute» isHeightAuto : EBoolean
«eAttribute» row : EInt
«eAttribute» column : EInt
«eAttribute» rowSpan : EInt
«eAttribute» columnSpan : EInt
«eAttribute» style : EString
«eAttribute» styleClassName : EString
«eClass» SLRadioButton
«eAttribute» groupName : EString
«eAttribute» isChecked : EBoolean
«eAttribute» isThreeState : EBoolean
«eAttribute» clickMode : ClickMode
«eClass» SLLabel
«eClass» SLSimpleContainer
«eAttribute» horizontalContentAlignment : HorizontalAlignment
«eAttribute» verticalContentAlignment : VerticalAlignment
«eClass» SLDataGrid
«eAttribute» gridLinesVisibility : EShort
«eAttribute» headersVisibility : EShort
«eAttribute» selectionMode : SLDataGridSelectionMode
«eClass» SLBorder
«eAttribute» paddingTop : EFloat
«eAttribute» paddingLeft : EFloat
«eAttribute» paddingRight : EFloat
«eAttribute» paddingBottom : EFloat
«eAttribute» cornerRadius : EFloat
«eClass» Widget
«eAttribute» name : EString
«eAttribute» isCoordinable : EBoolean
«eAttribute» isHidden : EBoolean
«eAttribute» isEnabled : EBoolean
«eAttribute» posX : EShort
«eAttribute» posY : EShort
«eAttribute» width : EShort
«eAttribute» height : EShort
«eAttribute» cardinality : EString
«eAttribute» uipattern : UIPattern
«eAttribute» customUIPattern : EString
«eClass» SLCheckBox
«eAttribute» isChecked : EBoolean
«eAttribute» isThreeState : EBoolean
«eAttribute» clickMode : ClickMode
«eClass» SLHyperlinkButton
«eAttribute» navigateUri : EString
«eClass» SLButton
«eAttribute» clickMode : ClickMode
«eClass» SLCustomCodeProperty
«eAttribute» name : EString
«eAttribute» type
«eAttribute» value : EString
+ customProperties *
1
«eClass» SLCustomCode
«eAttribute» placeholderPath : EString
«eClass» SLGridColumn
«eAttribute» width : EString
+ columns *
1
«eClass» SLGrid
1
«eClass» SLGridRow
«eAttribute» height : EString
+ rows *
«eClass» ScreenShot
«eAttribute» name : EString
«eAttribute» isEntry : EBoolean
«eClass» SLUserControl
+ userControl 0..1
«eClass» SLRegion
+ region 1
1
+ firstUserControl 0..1
1
+ usercontrols *
«eClass» SLUserControlInstance
«eClass» SLAccordionItem
«eAttribute» header : EString
«eAttribute» isSelected : EBoolean
+ items *
+ accordion 1
«eClass» SLAccordion
«eAttribute» expandDirection : EShort
«eAttribute» selectionMode : EShort
«eAttribute» selectionSecuence : EShort

Figure E.15. Silverlight components of the EMOF Extended Presentation Metamodel (part I).

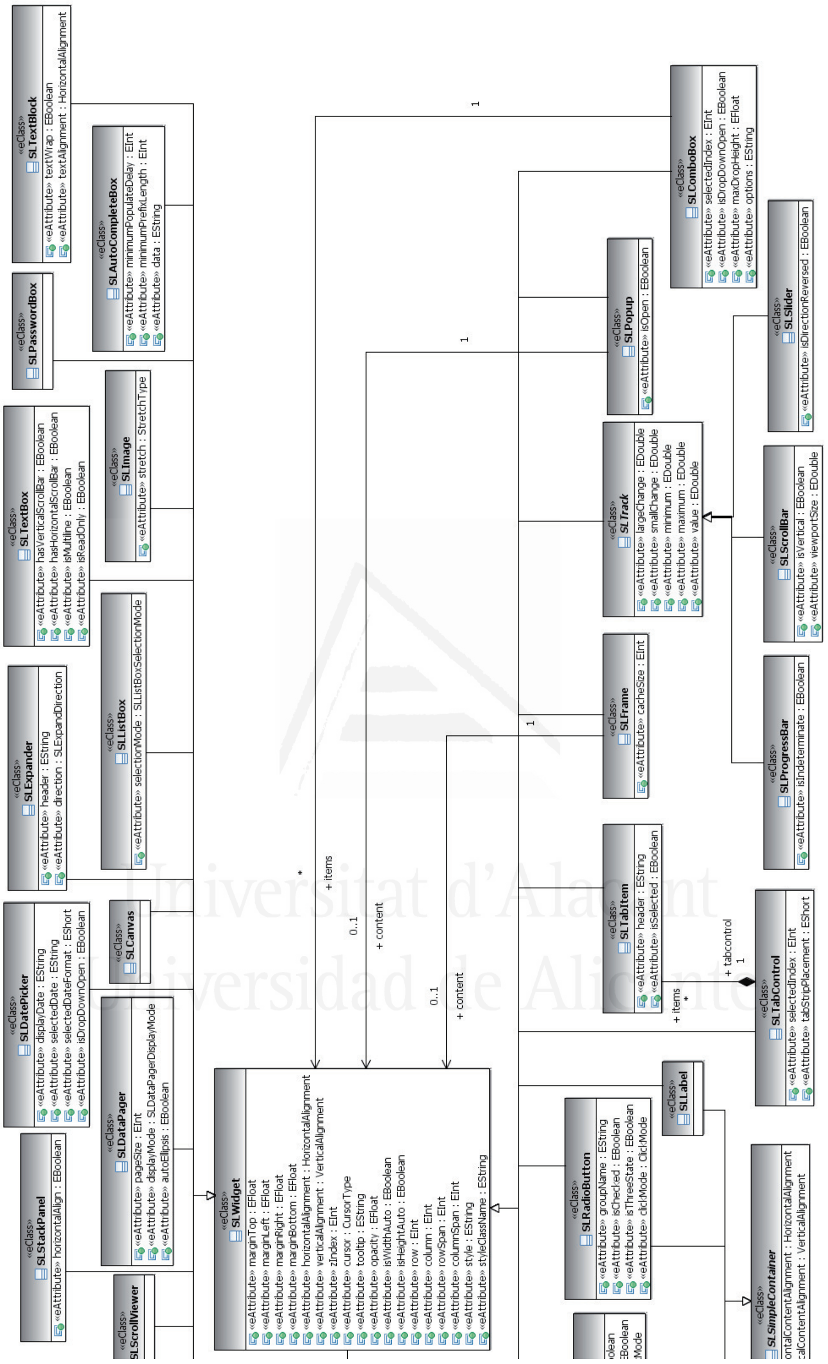

«eClass» SLTextBlock
«eAttribute» textWrap : EBoolean
«eAttribute» textAlignment : HorizontalAlignment
«eClass» SLPasswordBox
«eClass» SLAutoCompleteBox
«eAttribute» minimumPopulateDelay : EInt
«eAttribute» minimumPrefixLength : EInt
«eAttribute» data : EString
«eClass» SLTextBox
«eAttribute» hasVerticalScrollBar : EBoolean
«eAttribute» hasHorizontalScrollBar : EBoolean
«eAttribute» isMultiline : EBoolean
«eAttribute» isReadOnly : EBoolean
«eClass» SLImage
«eAttribute» stretch : StretchType
«eClass» SLExpander
«eAttribute» header : EString
«eAttribute» direction : SLExpandDirection
«eClass» SLListBox
«eAttribute» selectionMode : SLListBoxSelectionMode
«eClass» SLDatePicker
«eAttribute» displayDate : EString
«eAttribute» selectedDate : EString
«eAttribute» selectedDateFormat : EShort
«eAttribute» isDropDownOpen : EBoolean
«eClass» SLCanvas
«eClass» SLDataPager
«eAttribute» pageSize : EInt
«eAttribute» displayMode : SLDataPagerDisplayMode
«eAttribute» autoEllipsis : EBoolean
«eClass» SLStackPanel
«eAttribute» horizontalAlign : EBoolean
«eClass» SLScrollViewer
«eClass» SLWidget
«eAttribute» marginTop : EFloat
«eAttribute» marginLeft : EFloat
«eAttribute» marginRight : EFloat
«eAttribute» marginBottom : EFloat
«eAttribute» horizontalAlignment : HorizontalAlignment
«eAttribute» verticalAlignment : VerticalAlignment
«eAttribute» zIndex : EInt
«eAttribute» cursor : CursorType
«eAttribute» tooltip : EString
«eAttribute» opacity : EFloat
«eAttribute» isWidthAuto : EBoolean
«eAttribute» isHeightAuto : EBoolean
«eAttribute» row : EInt
«eAttribute» column : EInt
«eAttribute» rowSpan : EInt
«eAttribute» columnSpan : EInt
«eAttribute» style : EString
«eAttribute» styleClassName : EString
+ items
*
0..1
+ content
0..1
+ content
1
1
1
«eClass» SLComboBox
«eAttribute» selectedIndex : EInt
«eAttribute» isDropDownOpen : EBoolean
«eAttribute» maxDropHeight : EFloat
«eAttribute» options : EString
«eClass» SLPopup
«eAttribute» isOpen : EBoolean
«eClass» SLSlider
«eAttribute» isDirectionReversed : EBoolean
«eClass» SLTrack
«eAttribute» largeChange : EDouble
«eAttribute» smallChange : EDouble
«eAttribute» minimum : EDouble
«eAttribute» maximum : EDouble
«eAttribute» value : EDouble
«eClass» SLScrollBar
«eAttribute» isVertical : EBoolean
«eAttribute» viewportSize : EDouble
«eClass» SLFrame
«eAttribute» cacheSize : EInt
«eClass» SLProgressBar
«eAttribute» isIndeterminate : EBoolean
«eClass» SLTabItem
«eAttribute» header : EString
«eAttribute» isSelected : EBoolean
+ tabcontrol
1
+ items
*
«eClass» SLTabControl
«eAttribute» selectedIndex : EInt
«eAttribute» tabStripPlacement : EShort
«eClass» SLLabel
«eClass» SLRadioButton
«eAttribute» groupName : EString
«eAttribute» isChecked : EBoolean
«eAttribute» isThreeState : EBoolean
«eAttribute» clickMode : ClickMode
«eClass» SLSimpleContainer
ontalContentAlignment : HorizontalAlignment
calContentAlignment : VerticalAlignment

Figure E.16. Silverlight components of the EMOF Extended Presentation Metamodel (part II).

Annex F. TRANSFORMATION RULES

This annex introduces the code of transformation rules that could not be described in Chapter 7. Some parts of the code, reused from the OOH4RIA methodology, might have been omitted.

F.1. MODEL-TO-TEXT TRANSFORMATION RULES IN XPAND

This section presents the Xpand code of the following model-to-text transformation rules: *Bec_root* (Table F.1), *Adapter_root* (Table F.2), *Dto_root* (Table F.3) and *Service_root* (Table F.4).

Table F.1. Xpand code of the Bec_root model-to-text transformation rule.

```
«DEFINE Bec_root FOR ENModel-»
«FOREACH ((List[NavigationalClass])
navigationalElem.typeSelect(ExternalLink).nodeOrigin).referToClass AS class-»
«FILE class.getBECDirectory()-»

using System;
using System.Net;
using System.Collections.Generic;

using «class.getENPackage()-»;
using «class.getDACPackage()-»;
using System.Text.RegularExpressions;
using Newtonsoft.Json.Linq;

namespace «class.getBECPackage()-»
{
    public partial class «class.formattedClassName("BEC")-»
    {
        «EXPAND MethodMap FOREACH
navigationalElem.typeSelect(ExternalLink).select(e| ((NavigationalClass)
e.nodeOrigin).referToClass == class)-»
    }
}

«ENDFILE-»
«ENDFOREACH-»
«ENDDEFINE»

«DEFINE MethodMap FOR ExternalLink-»
«ENDDEFINE»

«DEFINE MethodMap FOR ExternalTraversalLink-»
    «LET ((NavigationalClass) this.nodeTarget).referToClass AS class-»
    «LET ((NavigationalClass)
this.nodeTarget).referToClass.formattedClassName("EN") AS enClass-»
        public IList<«enClass-»> «this.name.toFirstUpper()-»()
        {
            IList<«enClass-»> list = null;

            LinkedDataAccess.DAC.I«class.formattedClassName("EDAC")-»
«class.name.toLowerCase()-»DAC = new
LinkedDataAccess.DAC.«this.source.name.toFirstUpper()-
»«class.formattedClassName(getDACSuffix())-» ();

            list = «class.name.toLowerCase()-»DAC.«this.name.toFirstUpper()-»();
        }

    «ENDLET»
    «ENDLET»
«ENDDEFINE»
```

Table F.2. Xpand code of the Adapter_root model-to-text transformation rule.

```
«DEFINE Adapter_root FOR ENModel»
    «EXPAND netClassDTO(this) FOREACH ((List[ExtNavigationalClass])
this.navigationalElem.select(e|e.metaType == ExtNavigationalClass)).edmConcept -»
«ENDDEFINE»

«DEFINE netClassDTO(ENModel navigationalModel) FOR Concept»
«LET name.toFirstUpper() AS className-»
«FILE ((String)GLOBALVAR project) + "_" + navigationalModel.name + "WCF" +
fileSeparator() + "Adapters" + fileSeparator() + "LinkedDataManagement" +
fileSeparator() + this.model.name.toFirstUpper() + fileSeparator() +
this.name.toFirstUpper() + "Adapter.cs"»
using System;
using «getEntityPackage()»;
using «getDTOClassPackage()»;

namespace «getAdapterClassPackage()»
{
    public class «className»Adapter
    {
        public static «className»DTO Convert(«className»Entity en)
        {
            «className»DTO newinstance = null;

            if (en != null)
            {
                newinstance = new «className»DTO();

                if (en.__Uri != null)
                    newinstance.__Uri = en.__Uri;

                «IF this.domainClass != null-»
                «LET this.domainClass AS class»
                «IF class.identifiers().size > 1-»

newinstance.«class.getAllFathersRoot().get(0).formattedName().toFirstUpper()+"DTO_O
ID"» = en.«class.getAllFathersRoot().get(0).formattedName().toFirstUpper()+"OID"»;
                «ENDIF»
                «FOREACH class.navigableAttibutesAll() AS f-»
                    «IF (class.identifiers().size==1 && f.isOID) || !f.isOID-»
                        «IF f.association() != null-»
                            if (en.«f.formattedName().toFirstUpper()» != null)
                            {
                                «IF f.upper.isGreaterThanOne()-»

newinstance.«f.formattedName().toFirstUpper()»_oid = new
«f.dtoDataCreationType("DTO")»();
                                    foreach
(«projectName().toFirstUpper()»NHibernate.EN.«f.associationOtherSide().class.netPac
kageDomain()».«f.associationOtherSide().class.formattedName().toFirstUpper()»EN
entry in en.«f.formattedName().toFirstUpper()»)

newinstance.«f.formattedName().toFirstUpper()»_oid.Add(entry.«f.associationOtherSid
e().class.getOIDProperty().toFirstUpper()»);
                                «ELSE-»

newinstance.«f.formattedName().toFirstUpper()»_oid =
en.«f.formattedName().toFirstUpper()».«f.associationOtherSide().class.getOIDPropert
y().toFirstUpper()»;
                                «ENDIF-»
                            }
                        «ELSE-»
                            newinstance.«f.formattedName().toFirstUpper()» =
en.«f.formattedName().toFirstUpper()»;
                        «ENDIF-»
                    «ENDIF-»
                «ENDFOREACH-»
                «ENDLET-»
                «ELSE-»
                «FOREACH this.attributes() AS attr-»
                newinstance.«attr.name.toFirstUpper()» =
en.«attr.name.toFirstUpper()»;
                «ENDFOREACH-»
                «ENDIF-»
            }

            return newinstance;
        }
    }

}
    «ENDFILE-»
    «ENDLET-»
«ENDDEFINE»
```

Table F.3. Xpand code for the Dto_root model-to-text transformation rule.

```
«DEFINE Dto_root FOR ENModel-»
    «EXPAND DtoClasses(this) FOREACH
this.navigationalElem.typeSelect(ExtNavigationalClass).edmConcept-»
«ENDDEFINE»

«DEFINE DtoClasses(ENModel navigationalModel) FOR Concept-»
«LET this.name.toFirstUpper() + "DTO" AS csClassName-»
«FILE ((String)GLOBALVAR project) + "_" + navigationalModel.name + "WCF" +
fileSeparator() + "DTO" + fileSeparator() + "LinkedDataManagement" +
fileSeparator() + this.model.name.toFirstUpper() + fileSeparator() +
this.name.toFirstUpper() + "DTO.cs"-»
«REM»«FILE ((String)GLOBALVAR project) + "_LinkedDataCommon" + fileSeparator() +
"DTO" + fileSeparator() + this.model.name.toFirstUpper() + fileSeparator() +
csClassName + ".cs"-»«ENDREM»
using System;
using System.Runtime.Serialization;

namespace «this.getDTOClassPackage()»
{
    [DataContract(Name = "«csClassName»")]
    public class «csClassName» «IF this.domainClass != null-»:
«this.domainClass.getDTOPackage()».«this.domainClass.formattedClassName("DTO")»«END
IF»
    {
        private string __uri = "";
        [DataMember]
        public string __Uri { get{ return this.__uri; } set{ this.__uri = value; }
}

        «IF this.domainClass == null-»
        «FOREACH this.attributes() AS attr-»
        «LET attr.getCsType() AS type-»
        «LET attr.name.toFirstLower() AS attrName-»
        private «type» «attrName»;
        [DataMember]
        public «type» «attrName.toFirstUpper()»{ get{ return this.«attrName»; }
set{ this.«attrName» = value; } }
```

```
        «ENDLET-»
        «ENDLET-»
        «ENDFOREACH-»
        «ENDIF-»
    }
}
«ENDFILE»
«ENDLET-»
«ENDDEFINE»
```

Table F.4. Xpand code of the Service_root model-to-text transformation rule.

```
«DEFINE Service_root FOR ENModel»
«FILE projectName().toFirstUpper()+"_"+((NavigationalModel) GLOBALVAR
enModel).formattedName()+"WCF"+fileSeparator()+"ServiceExternal.svc.cs"-»
using System;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Activation;
using NHibernate;
using LinkedDataManagement.«this.edModel.name.toFirstUpper()».Gateways;

namespace «getEWCFPackage()»
{
    public partial class Service
    {
        «EXPAND ServiciesMainImpl-»
        «PROTECT CSTART '/*' BECD '*/' ID
        ((String)GLOBALVAR project) + "_" + ((NavigationalModel) GLOBALVAR
enModel).formattedName()
        + "WCF_External_Other_Operations"»

        «ENDPROTECT»
    }
}
«ENDFILE»
«ENDDEFINE»

«DEFINE ServiciesMainImpl FOR ENModel-»
    «EXPAND ServiceImpl FOREACH
(List[NavigationalClass])this.navigationalElem.select(e |
NavigationalClass.isInstance(e))-»
    «REM»«EXPAND ServiciesMainImpl FOREACH packages()»«ENDREM»
«ENDDEFINE»

«DEFINE ServiceImpl FOR NavigationalClass-»
«REM»
«FOREACH operations() AS o-»
[OperationContract]
«IF o.referToOperation.operationType==OperationType::Custom-»
    «o.referToOperation.visibility.toString().toLowerCase()» «EXPAND CustomMethod
FOR o»
{
    «EXPAND declareBodyCustom FOR o.referToOperation-»
}

«ELSEIF o.referToOperation.operationType==OperationType::ReadOID-»
«o.referToOperation.visibility.toString().toFirstLower()-» «EXPAND ReadOIDMethod
FOR o»
{
    «EXPAND ReadOIDBody FOR o.referToOperation-»
}

«ELSEIF o.referToOperation.operationType==OperationType::ReadAll-»
«FOREACH o.slEnables() AS s-»
«o.referToOperation.visibility.toString().toFirstLower()» «EXPAND ReadAllMethod FOR
```

```
s»
{
    «EXPAND ReadAllBodySer FOR s-»
}
«ENDFOREACH-»

«ELSEIF o.referToOperation.operationType==OperationType::ReadFilter-»
«FOREACH o.slEnables() AS s-»
«o.referToOperation.visibility.toString().toFirstLower()» «EXPAND ReadFilterMethod
FOR s»
{
    «EXPAND ReadFilterBody(s) FOR o.referToOperation»-
}
«ENDFOREACH-»
«ENDIF-»
«ENDFOREACH-»
«ENDREM»
«FOREACH this.nm.navigationalElem.typeSelect(TravesalLink).select(e|e.nodeOrigin ==
this && e.nodeTarget.metaType == ExtNavigationalClass) AS t-»
«LET ((ExtNavigationalClass) t.nodeTarget).edmConcept AS class-»
«LET class.getDTOClassPackage() AS packageName-»
«LET class.name.toFirstUpper() + "DTO" AS dtoName-»
«LET class.getEntityPackage() AS enPackageName-»
«LET class.name.toFirstUpper() + "Entity" AS enName-»
[OperationContract]
public System.Collections.Generic.IList<«packageName».«dtoName»> «t.nameMethod()»
(«EXPAND TraversalLinkArguments FOR t-»)
{
    System.Collections.Generic.IList<«packageName».«dtoName»> dto = null;
    System.Collections.Generic.IList<«enPackageName».«enName»> en = null;

    try
    {
        var gateway = new «((ExtNavigationalClass)
t.nodeTarget).source.name.toFirstUpper()»ServiceGateway();

        en = gateway.«
        t.formattedName().toFirstUpper()»_«((NavigationalModel) GLOBALVAR
enModel).formattedName()» («IF t.paging»offset, limit«ENDIF»«IF t.metaType ==
ExternalTraversalLink && ((ExternalTraversalLink)t).edmObjectProperty != null»«IF
t.paging», «ENDIF»uri«ENDIF»);

        if (en != null)
        {
            dto = new System.Collections.Generic.List<«packageName».«dtoName»>();

            foreach («enPackageName».«enName» item in en)
            {

dto.Add(«class.getAdapterClassPackage()».«class.name.toFirstUpper()»Adapter.Convert
(item));
            }
        }
    }
    «EXPAND catch»
    return dto;
}
«ENDLET-»
«ENDLET-»
«ENDLET-»
«ENDLET-»
«ENDLET-»
«ENDFOREACH-»
«ENDDEFINE»

«DEFINE TraversalLinkArguments FOR TravesalLink-»
«ENDDEFINE»

«DEFINE CustomMethod FOR NavigationalOperation-»
    «IF referToOperation.type==PrimitiveType::Object &&
referToOperation.collectionType==CollectionType::None»
    «referToOperation.typeObject.netPackageDomainProject("_"+((NavigationalModel)
GLOBALVAR
enModel).formattedName()+"WCF","DTO")».«referToOperation.typeObject.formattedClassN
ame("DTO")»«
    ELSEIF referToOperation.type==PrimitiveType::Object &&
referToOperation.collectionType!=CollectionType::None-»

«referToOperation.collectionType.interfaceCollectionType()»<«referToOperation.typeO
bject.netPackageDomainProject("_"+((NavigationalModel) GLOBALVAR
enModel).formattedName()+"WCF","DTO")».«
    referToOperation.typeObject.formattedClassName("DTO")»>
    «ELSE-»
    «referToOperation.dataType("DTO")»
    «ENDIF-»
    «nameMethod()» («FOREACH referToOperation.arguments AS a SEPARATOR ','-»
            «a.dataType("DTO")+" "+a.formattedName().toFirstLower()-»
    «ENDFOREACH-»)
«ENDDEFINE»

«DEFINE ReadOIDMethod FOR Operation-»
«ENDDEFINE»

«DEFINE ReadOIDMethod FOR NavigationalOperation-»
«referToOperation.class.netPackageDomainProject("_"+((NavigationalModel) GLOBALVAR
enModel).formattedName()+"WCF","DTO")».«referToOperation.class.formattedClassName("
DTO")» «nameMethod()-»
(«FOREACH referToOperation.arguments AS arg»«arg.dataTypeOID()»
«arg.formattedName()»«ENDFOREACH-»)«
ENDDEFINE»


«DEFINE ReadAllMethod FOR Operation-»
«ENDDEFINE»

«DEFINE ReadAllMethod FOR ServiceLink-»
«ENDDEFINE»

«DEFINE ReadFilterMethod FOR Operation-»
«ENDDEFINE»

«DEFINE ReadFilterMethod FOR ServiceLink-»
«ENDDEFINE»

«DEFINE NewMethod FOR Operation-»
    «class.dataTypeOID()» «nameMethod()-
    »(«FOREACH arguments AS a SEPARATOR ','-»
            «a.dataType("DTO")+" "+a.formattedName().toFirstLower()-»
    «ENDFOREACH-»)«
ENDDEFINE»

«DEFINE NewMethod FOR NavigationalOperation-»
    «class.referToClass.dataTypeOID()» «nameMethod()-
    »(«FOREACH referToOperation.arguments AS a SEPARATOR ','-»
            «a.dataType("DTO")+" "+a.formattedName().toFirstLower()-»
    «ENDFOREACH-»)«
ENDDEFINE»

«DEFINE ModifDestRelaUnrelMethod FOR Operation-»
ENDDEFINE»

«DEFINE ModifDestRelaUnrelMethod FOR NavigationalOperation-»
ENDDEFINE»

«DEFINE catch FOR NavigationalClass»
    catch(Exception ex)
    {
        throw ex;
    }
«ENDDEFINE»
```

```
«DEFINE catch FOR Operation»
«ENDDEFINE»

«DEFINE declareBodyCustom FOR Operation»
    «EXPAND declareBEC»
    «EXPAND declareDAC»
    «IF type==PrimitiveType::Object && collectionType==CollectionType::None»
        «typeObject.netPackageDomainProject()».«
        typeObject.formattedName().toFirstUpper()»EN returnValueEN=null;
        «class.netPackageDomainProject("_"+((NavigationalModel) GLOBALVAR
enModel).formattedName()+"WCF","DTO")».«
        typeObject.formattedName().toFirstUpper()»DTO returnValueDTO=null;
    «ELSEIF type==PrimitiveType::Object && collectionType!=CollectionType::None»

«collectionType.interfaceCollectionType()»<«typeObject.netPackageDomainProject("NHi
bernate","EN")».«
        typeObject.formattedClassName("EN")»> returnValueEN=null;

«collectionType.interfaceCollectionType()»<«typeObject.netPackageDomainProject("_"+
((NavigationalModel) GLOBALVAR enModel).formattedName()+"WCF","DTO")».«
        typeObject.formattedClassName("DTO")»> returnValueDTO=null;
        «ELSE-»
        «IF dataType("").compareTo("void")!=0»
            «dataType("")» returnValueEN;
        «ENDIF»
    «ENDIF»
    try
    {
        using(ISession session=NHibernateHelper.OpenSession())
           using (ITransaction tx = session.BeginTransaction())
        {
        «EXPAND inicialiceDAC»
        «EXPAND inicialiceBEC»
        «EXPAND dto2EN»
        «IF dataType("").compareTo("void")!=0»returnValueEN=«ENDIF-»

«class.formattedClassName("BEC").toFirstLower()».«formattedName().toFirstUpper()»(«
        EXPAND argumentsDTO2argumentsEN»);
        «IF type==PrimitiveType::Object && collectionType==CollectionType::None»
            returnValueDTO=«class.netPackageDomainProject("_"+((NavigationalModel)
GLOBALVAR enModel).formattedName()+"WCF","Adapters")».«

typeObject.formattedName().toFirstUpper()»Adapter.Convert(returnValueEN);
        «ELSEIF type==PrimitiveType::Object &&
collectionType!=CollectionType::None»
        if(returnValueEN!=null)
        {
            returnValueDTO=new
«collectionType.collectionTypeImp()»<«typeObject.netPackageDomainProject("_"+((Navi
gationalModel) GLOBALVAR enModel).formattedName()+"WCF","DTO")».«
        typeObject.formattedClassName("DTO")»>();

foreach(«typeObject.netPackageDomainProject()+"."+typeObject.formattedClassName("EN
")» item in
                returnValueEN){

returnValueDTO.Add(«typeObject.netPackageDomainProject("_"+((NavigationalModel)
GLOBALVAR enModel).formattedName()+"WCF","Adapters")».«
                typeObject.formattedName().toFirstUpper()»Adapter.Convert(item));
            }
        }
        «ENDIF»
        «EXPAND commitSession»
        }
    }
    «EXPAND catch»
    «IF type==PrimitiveType::Object»
    return returnValueDTO;
    «ELSEIF dataType("").compareTo("void")!=0»
    return returnValueEN;
    «ENDIF»
«ENDDEFINE»

«DEFINE ReadFilterBody(ServiceLink serviceLink) FOR Operation»
«IF this.collectionType == CollectionType::None-»
    «EXPAND declareDTO»
«ELSE-»
    «EXPAND declareCollectionDTO»
«ENDIF-»
«IF this.collectionType == CollectionType::None-»
    «EXPAND declareEN-»
«ELSE-»
    «EXPAND declareCollectionEN»
«ENDIF-»
«EXPAND declareBEC»
«EXPAND declareDAC»
try
{
        using(ISession session=NHibernateHelper.OpenSession())
           using (ITransaction tx = session.BeginTransaction())
           {
        «IF this.collectionType == CollectionType::None-»
            «EXPAND inicialiceDTO»
        «ELSE-»
            «EXPAND inicialiceCollectionDTO»
        «ENDIF-»
        «IF this.collectionType == CollectionType::None-»
            «EXPAND inicialiceEN»
        «ELSE-»
            «EXPAND inicialiceCollectionEN»
        «ENDIF-»
        «EXPAND inicialiceDAC»
        «EXPAND inicialiceBEC»
        «this.typeObject.formattedName().toFirstLower()»EN«IF this.collectionType
!= CollectionType::None»s«ENDIF»=«

class.formattedClassName("BEC").toFirstLower()».«formattedName().toFirstUpper()-»
        («FOREACH arguments AS arg SEPARATOR
','»«arg.formattedName()»«ENDFOREACH»«IF serviceLink.paging == true»«IF
serviceLink.argumentLink.size > 0»,«ENDIF»first, «serviceLink.chunkSize»«ELSEIF
this.paging == true»«IF serviceLink.argumentLink.size > 0»,«ENDIF»0, -1«ENDIF»);
        «IF this.collectionType == CollectionType::None-»

«this.typeObject.formattedName().toFirstLower()»DTO=«this.typeObject.netPackageDoma
inProject("_"+((NavigationalModel) GLOBALVAR enModel).name+"WCF","Adapters")».«

this.typeObject.formattedName().toFirstUpper()»Adapter.Convert(«this.typeObject.for
mattedName().toFirstLower()»EN);
        «ELSE-»
        if(«this.typeObject.formattedName().toFirstLower()»ENs!=null)
        {

foreach(«this.typeObject.netPackageDomainProject()+"."+this.typeObject.formattedCla
ssName("EN")» item in «
            this.typeObject.formattedName().toFirstLower()»ENs){

«this.typeObject.formattedName().toFirstLower()»DTOs.Add(«this.typeObject.netPackag
eDomainProject("_"+((NavigationalModel) GLOBALVAR
enModel).name+"WCF","Adapters")».«

this.typeObject.formattedName().toFirstUpper()»Adapter.Convert(item));
            }
        }
        «ENDIF»
        «EXPAND commitSession»
        }
}
    «EXPAND catch»
    return «this.typeObject.formattedName().toFirstLower()»DTO«IF
```

```
this.collectionType != CollectionType::None»s«ENDIF»;
«ENDDEFINE»

«DEFINE ReadFilterBody FOR Operation»
«ENDDEFINE»

«DEFINE ReadAllBody FOR Operation»
«EXPAND declareCollectionDTO»
«EXPAND declareCollectionEN»
«EXPAND declareDAC»
«EXPAND declareBEC»
try
{
    using(ISession session=NHibernateHelper.OpenSession())
       using (ITransaction tx = session.BeginTransaction())
    {
        «EXPAND inicialiceCollectionDTO»
        «EXPAND inicialiceCollectionEN»
        «EXPAND inicialiceDAC»
        «EXPAND inicialiceBEC»

«class.formattedName().toFirstLower()»ENs=«class.formattedClassName("BEC").toFirstL
ower()».«
        formattedName().toFirstUpper()»(0,-1);
        if(«class.formattedName().toFirstLower()»ENs!=null)
        {

foreach(«class.netPackageDomainProject()+"."+class.formattedClassName("EN")» item
in «
            class.formattedName().toFirstLower()»ENs){

«class.formattedName().toFirstLower()»DTOs.Add(«class.netPackageDomainProject("_"+(
(NavigationalModel) GLOBALVAR enModel).formattedName()+"WCF","Adapters")».«
                class.formattedName().toFirstUpper()»Adapter.Convert(item));
            }
        }
        «EXPAND commitSession»
    }
}
«EXPAND catch»
return «class.formattedName().toFirstLower()»DTOs;
«ENDDEFINE»

«DEFINE ReadAllBodySer FOR ServiceLink»
«ENDDEFINE»

«DEFINE ReadOIDBody FOR Operation»
«EXPAND declareDTO»
«EXPAND declareEN»
«EXPAND declareBEC»
«EXPAND declareDAC»
try
{
    using(ISession session=NHibernateHelper.OpenSession())
       using (ITransaction tx = session.BeginTransaction())
    {
    «EXPAND inicialiceDTO»
    «EXPAND inicialiceEN»
    «EXPAND inicialiceDAC»
    «EXPAND inicialiceBEC»

«class.formattedName().toFirstLower()»EN=«class.formattedClassName("BEC").toFirstLo
wer()».«
    formattedName().toFirstUpper()
    »(«FOREACH arguments AS arg SEPARATOR ','»«arg.formattedName()»«ENDFOREACH»);

«class.formattedName().toFirstLower()»DTO=«class.netPackageDomainProject("_"+((Navi
gationalModel) GLOBALVAR enModel).formattedName()+"WCF","Adapters")».«

class.formattedName().toFirstUpper()»Adapter.Convert(«class.formattedName().toFirst
Lower()»EN);
    «EXPAND commitSession»
    }
}
«EXPAND catch»
return «class.formattedName().toFirstLower()»DTO;
«ENDDEFINE»

«DEFINE dto2EN FOR Operation-»
        «FOREACH arguments AS arg-»
            «IF arg.type==PrimitiveType::Object &&
arg.collectionType==CollectionType::None-»
                «arg.dataType("EN")» «arg.formattedName()»EN =
«arg.typeObject.netPackageDomainProject("_"+((NavigationalModel) GLOBALVAR
enModel).formattedName()+"WCF","AdaptersEN")».«

arg.typeObject.formattedClassName("DTOENAdapter")».Convert(«arg.formattedName()»);
            «ELSEIF arg.type==PrimitiveType::Object &&
arg.collectionType!=CollectionType::None-»
                «arg.dataType("EN")» «arg.formattedName()»EN=new
«arg.collectionType.collectionTypeImp()»<«

arg.typeObject.netPackageDomainProject()».«arg.typeObject.formattedClassName("EN")»
>();

foreach(«arg.typeObject.netPackageDomainProject("_"+((NavigationalModel) GLOBALVAR
enModel).formattedName()+"WCF","DTO")».«arg.typeObject.formattedClassName("DTO")»
aux in «
                arg.formattedName()»)
                {

«arg.formattedName()»EN.Add(«arg.typeObject.netPackageDomainProject("_"+((Navigatio
nalModel) GLOBALVAR enModel).formattedName()+"WCF","AdaptersEN")».«
                arg.typeObject.formattedClassName("DTOENAdapter")».Convert(aux));
                }
            «ENDIF-»
        «ENDFOREACH-»«
        ENDDEFINE»

«DEFINE argumentsDTO2argumentsEN FOR Operation-»
«FOREACH arguments AS arg SEPARATOR ','-»
        «IF arg.type==PrimitiveType::Object-»
        «arg.formattedName().toFirstLower()»EN«
        ELSE-»
        «arg.formattedName().toFirstLower()»«ENDIF-»
        «ENDFOREACH-»
«ENDDEFINE»

«DEFINE declareBEC FOR Operation»
«class.netPackageDomainProject("BEC")».«
class.formattedClassName("BEC")»
«class.formattedClassName("BEC").toFirstLower()»=null;«
ENDDEFINE»

«DEFINE inicialiceBEC FOR Operation-»
«class.formattedClassName("BEC").toFirstLower()
»=new
«class.netPackageDomainProject("BEC")».«class.formattedClassName("BEC")»(_I«class.f
ormattedClassName("DAC").toFirstLower()»);«
ENDDEFINE»


«DEFINE declareDAC FOR Operation-»
«ENDDEFINE»

«DEFINE inicialiceDAC FOR Operation-»
«ENDDEFINE»

«DEFINE declareEN FOR Operation»
«ENDDEFINE»
```

```
«DEFINE inicialiceEN FOR Operation»
«ENDDEFINE»

«DEFINE declareCollectionEN FOR Operation-»
    «IF this.operationType == OperationType::ReadFilter-»

System.Collections.Generic.IList<«this.typeObject.netPackageDomainProject()+"."+thi
s.typeObject.formattedClassName("EN")»>
«this.typeObject.formattedName().toFirstLower()»ENs=null;«
    ELSE-»

System.Collections.Generic.IList<«class.netPackageDomainProject()+"."+class.formatt
edClassName("EN")»> «class.formattedName().toFirstLower()»ENs=null;«
    ENDIF-»
«ENDDEFINE»

«DEFINE inicialiceCollectionEN FOR Operation-»
    «IF this.operationType == OperationType::ReadFilter-»
        «this.typeObject.formattedName().toFirstLower()»ENs=new
System.Collections.Generic.List<«

this.typeObject.netPackageDomainProject()+"."+this.typeObject.formattedClassName("E
N")»>();«
    ELSE»
        «class.formattedName().toFirstLower()»ENs=new
System.Collections.Generic.List<«
        class.netPackageDomainProject()+"."+class.formattedClassName("EN")»>();«
    ENDIF-»
«ENDDEFINE»

«DEFINE declareDTO FOR Operation»
«class.netPackageDomainProject("_"+((NavigationalModel) GLOBALVAR
enModel).formattedName()+"WCF","DTO")».«class.formattedClassName("DTO")»
«class.formattedName().toFirstLower()»DTO=null;«
ENDDEFINE»

«DEFINE declareCollectionDTO FOR Operation-»
«ENDDEFINE»

«DEFINE inicialiceCollectionDTO FOR Operation-»
«ENDDEFINE»

«DEFINE inicialiceDTO FOR Operation»
«ENDDEFINE»

«DEFINE commitSession FOR Operation-»
tx.Commit();«
ENDDEFINE»
```

F.2. MODEL-TO-MODEL TRANSFORMATION RULES IN QVT OPERATIONAL

This section presents the QVTo code of the *Domain2Navigation* (Table F.5) and *Navigation2Presentation* (Table F.6) model-to-model transformations.

Table F.5. ***Domain2Navigation*** **model-to-model QVTo transformation.**

```
modeltype DOM uses "http://www.insidesoft.net/conceptualView/1.0.0";
modeltype NAV uses "http://www.insidesoft.net/navigationalView";
modeltype GMF uses "http://www.eclipse.org/gmf/runtime/1.0.2/notation";
modeltype ecore uses "http://www.eclipse.org/emf/2002/Ecore";

transformation Domain2Navigation(in inModel: DOM, out outModel: NAV, out
outDiagram: GMF);

main()
{
    inModel.rootObjects()[ConceptualModel].map Model2Model();
    outModel.rootObjects()[NavigationalModel].map Model2Diagram();
}

mapping ConceptualModel::Model2Model() : NavigationalModel
{
    result.name := self.name;
    result.conceptualModel := self;
    result.applicationFacade := ApplicationFacadeType::WCF;

    // Create the entry point
    var home := object NavigationalClass
        {
            name := "Home";
            isEntryPoint := true;

            referToClass := self.elements[Class]->select(c | c.name = "User" or
c.name = "Usuario")->asSequence()->first();
        };

    result.navigationalElem += home;

    result.navigationalElem += self.elements[Class]->map Class2NavClass();

    // Obtain those classes that are components of others
    var componentClasses := self.elements[Association]->select(a |
a.rolOrigin.aggregation = AggregationKind::Composite).classTarget->asSet();
    componentClasses := componentClasses->union(self.elements[Association]-
>select(a | a.rolTarget.aggregation = AggregationKind::Composite).classOrigin-
>asSet());

    // Create the links from the entry point to each of the Navigational classes
    result.navigationalElem[NavigationalClass]->select(c | not c.isEntryPoint and
not componentClasses->includes(c.referToClass))->forEach(navClass)
    {
        result.navigationalElem += object TravesalLink
        {
            name := "GetAll" + navClass.name.firstToUpper();

            isSameNode := true;

            targetNavigationPattern := AccessType::ShowAll;

            activationMode := ActivationType::Manual;

            nodeOrigin := home;

            nodeTarget := navClass;
        };
    };

    result.navigationalElem[NavigationalClass].navOperation->forEach(op)
    {
        op.sl := op.map NavOperation2ServiceLink();
        result.navigationalElem += op.sl;
    };
    self.elements[Association]->map Association2TraversalLink(result)-
>forEach(link)
    {
        result.navigationalElem += link;
    };

}

mapping NavigationalModel::Model2Diagram() : Diagram
{
    result.element := self.oclAsType(EObject);
    result.name := self.name;
    result.measurementUnit := MeasurementUnit::Pixel;
    result.type := "NavigationalView";
}

mapping Class::Class2NavClass() : NavigationalClass
{
    result.name := self.name;
    result.referToClass := self;

    result.navAttribute += self.getAscendants().attributes[associationOrigin = null
and associationTarget = null]->map Attribute2NavAttribute(result);

    result.navAttribute += self.attributes[associationOrigin = null and
associationTarget = null]->map Attribute2NavAttribute(result);

    result.navOperation += self.operations->xselect(e | e.operationType <>
OperationType::Custom)->map Operation2NavOperation(result);
}

mapping Attribute::Attribute2NavAttribute(in navClass: NavigationalClass) :
NavigationalAttribute
{
    result.name := self.name;

    result.referToAttribute := self;

    result._class := navClass;
}

mapping Operation::Operation2NavOperation(in navClass: NavigationalClass) :
NavigationalOperation
{
    result.name := self.name;

    result.referToOperation := self;

    result.alternativeArguments += self.arguments->map Argument2ArgumentLink();
}

mapping Argument::Argument2ArgumentLink() : ArgumentLink
{
    result.name := self.name;

    result.argument := self;

    result.value := self.value;
}

mapping NavigationalOperation::NavOperation2ServiceLink() : ServiceLink
{
    result.name := self.name + self._class.name.firstToUpper();

    result.operationName := self.name;

    result.nodeOrigin := self._class;

    result.nodeTarget := self._class;
    result.activationMode := ActivationType::Manual;

    result.targetNavigationPattern := AccessType::ShowAll;

    result.argumentLink += self.referToOperation.arguments->map
Argument2ArgumentLink();

    result.isSameNode := true;
}

mapping Association::Association2TraversalLink(in model : NavigationalModel) :
Sequence(TravesalLink)
{
    var originNodeAux := model.navigationalElem[NavigationalClass]->xselect(e | not
e.isEntryPoint and e.referToClass = self.classOrigin)->asSequence();
    var targetNodeAux := model.navigationalElem[NavigationalClass]->xselect(e | not
e.isEntryPoint and e.referToClass = self.classTarget)->asSequence();

    var originDescendants := originNodeAux->first().referToClass.getDescendants();
    var targetDescendants := targetNodeAux->first().referToClass.getDescendants();

    var originNodeList := model.navigationalElem[NavigationalClass]->xselect(e |
not e.isEntryPoint and originDescendants->includes(e.referToClass))-
>union(originNodeAux->asSet());
    var targetNodeList := model.navigationalElem[NavigationalClass]->xselect(e |
not e.isEntryPoint and targetDescendants->includes(e.referToClass))-
>union(targetNodeAux->asSet());

    if (self.rolOrigin.navigable)
    then
    {
        originNodeList->forEach(originNode)
        {
            targetNodeList->forEach(targetNode)
            {
                result += object TravesalLink
                    {
                        name := getTraversalLinkName(originNode.referToClass,
targetNode.referToClass, self.rolOriginMultiplicity);

                        associationRol := self.rolOrigin;

                        isSameNode := true;

                        targetNavigationPattern := AccessType::ShowAll;

                        if (self.rolOrigin.aggregation =
AggregationKind::Composite)
                        then
                            activationMode := ActivationType::Automatic
                        else
                            activationMode := ActivationType::Manual
                        endif;

                        nodeOrigin := originNode;

                        nodeTarget := targetNode;

                        nm := model;
                    };
            };
        };
    }
    endif;

    if (self.rolTarget.navigable)
    then
    {
        originNodeList->forEach(originNode)
        {
            targetNodeList->forEach(targetNode)
            {
                result += object TravesalLink
                    {
                        name := getTraversalLinkName(targetNode.referToClass, 
originNode.referToClass, self.rolTargetMultiplicity);

                        associationRol := self.rolTarget;

                        isSameNode := true;

                        targetNavigationPattern := AccessType::ShowAll;

                        if (self.rolTarget.aggregation = 
AggregationKind::Composite)
                        then
                            activationMode := ActivationType::Automatic
                        else
                            activationMode := ActivationType::Manual
                        endif;

                        nodeOrigin := targetNode;

                        nodeTarget := originNode;

                        nm := model;
                    }
            };
        };
    }
    endif;
}

/* Helpers */

helper getTraversalLinkName(in origin : Class, in target : Class, in multiplicity : 
String) : String {}

helper Class::getDescendants() : Sequence(Class) {}

helper Class::getAscendants() : Sequence(Class) {}
```

Table F.6. ***Navigation2Presentation*** **model-to-model QVTo transformation.**

```
import Presentation2Diagram;

modeltype DOM uses "http://www.insidesoft.net/conceptualView/1.0.0";
modeltype NAV uses "http://www.insidesoft.net/navigationalView";
modeltype SLPRES uses "http://www.insidesoft.net/silverlightPresentationView";
modeltype PRES uses "http://www.insidesoft.net/abstractPresentationView";
modeltype GMF uses "http://www.eclipse.org/gmf/runtime/1.0.2/notation";

transformation Navigation2Presentation(in inModel : NAV, out outModel : SLPRES, out 
outDiagram : GMF);

main()
{
    inModel.rootObjects()[NavigationalModel].map Model2Model();
    outModel.rootObjects()[SLPresentationModel].map Model2Diagram();
}

mapping NavigationalModel::Model2Model() : SLPresentationModel
{
    result.name := self.name;

    result.navigationalModel := self;

    result.height := 800;
    result.width := 1024;

    result.isWidthAuto := result.isHeightAuto := true;

    result.navContext := self.navigationalElem[NavigationalClass]->select(c |
c.isEntryPoint)->asSequence()->first();

    var rootGrid := object SLGrid
    {
        name := "RootGrid";
        height := 800;
        width := 1024;

        isEnabled := true;
        isHidden := false;

        style := 'Background="#ffffffff"';

        rowSpan := 1;
        columnSpan := 1;

        posX := 0;
        posY := 0;
    };

    result.referredWidgets += rootGrid;

    rootGrid.widgets += object SLStackPanel
    {
        name := ("StackPanel_" + self.name + "_item").getUniqueString();

        height := 800;
        width := 1024;

        widgets += object SLCanvas
        {
            name := "Canvas_header";
            height := 100;
            width := 1024;
        };

        widgets += object SLTabControl
        {
            name := "TabControl_menu";
            height := 700;
            width := 1024;

            String.restartAllStrCounter();

            items += self.navigationalElem[TravesalLink]->select(l |
l.nodeOrigin.isEntryPoint)->map TraversalLink2TabItem();
        };
    };
}

mapping TravesalLink::TraversalLink2TabItem() : SLTabItem
{
    name := "TabItem".getUniqueString();

    header := self.nodeTarget.name;

    height := 30;
    width := 60;

    style := getDefaultTextStyle();

    var stackPanel : SLStackPanel := object SLStackPanel
    {
        name := "SLStackPanel_container".getUniqueString();
        height := 660;
        width := 1010;

        horizontalAlign := true;
    };

    stackPanel.widgets += self.map TraversalLink2ItemList();

    // Retrieve the item list in order to update it with the new form
    var itemList := stackPanel.widgets->first().oclAsType(SLStackPanel).widgets-
>select(w | w.oclIsTypeOf(SLScrollViewer))->asSequence()-
>first().oclAsType(SLScrollViewer).widgets->first().oclAsType(SLStackPanel);

    stackPanel.widgets += object SLStackPanel
    {
        name := "SLStackPanel_detail".getUniqueString();

        width := 500;
        height := 500;

        var itemContainer := self.nodeTarget.oclAsType(NavigationalClass).map
NavClass2Item(itemList);
        // ADD ITEM
        widgets += itemContainer;

        // Link See Details button with the corresponding context
        var button : SLHyperlinkButton :=
itemList.FindWidget("HyperlinkButton_SeeDetails").oclAsType(SLHyperlinkButton);

        if (button <> null) then
        {
            var method :=
itemContainer.FindWidget("_item").oclAsType(SLStackPanel).methods->select(m |
m.name = "SetContentData")->asSequence()->first().oclAsType(WMethod);
            var onClickEvent := object WEvent{ name := "OnClick" };
            button.events += onClickEvent;
            button.ecas += object EventCall
            {
                event := onClickEvent;

                conditions += object Condition
                {
                    expresion := "true";

                    trueActions += object ActionCall
                    {
                        action := object ClientAction {    wMethod := method };

                        arguments += object ClientArgument
                        {
                            wMethodParameter := method.parameters->first();
                            value := "Context";
                        };
                    }
                }
            };
        }
        endif;

        // NEW FORM
        // Create form for new elements
        var newElementLinks :=
inModel.rootObjects()[NavigationalModel].navigationalElem[ServiceLink]
            ->select(l | l.nodeOrigin = self.nodeTarget and
l.navOperation.referToOperation.operationType = OperationType::New);

        if (newElementLinks->size() <> 0)
        then
        {
            var newForm := newElementLinks->asSequence()->first().map
ServiceLink2NewForm(null, itemList);
            newForm.marginTop := 10;

            var expander : SLExpander := addExpander(newForm);
            expander.header := "New " + newElementLinks->asSequence()-
>first().oclAsType(ServiceLink).nodeOrigin.name;

            widgets += expander;
        }
        endif;
    };

    var scrollViewer := addScrollViewer(stackPanel);
    scrollViewer.height := 660;
    scrollViewer.width := 1010;

    widgets += scrollViewer;
}

query SLStackPanel::FindWidget(in name : String) : SLWidget
{
    /* Auxiliary method */
}

mapping NavigationalClass::NavClass2ListItem(in container : SLStackPanel) :
SLStackPanel
{
    result.name := ("StackPanel_" + self.name + "_listItem").getUniqueString();

    result.isEnabled := true;
    result.isHidden := false;

    result.marginBottom := 2.0;

    result.style := 'Background="#ffaaaaaa"';

    var numElements := self.navAttribute->size();

    result.width := 500;
    result.height := 80;
    result.isHeightAuto := true;

    // ITEM FEATURES
    result.widgets += self.navAttribute->select(attr |
attr.referToAttribute.isOID)->NavAttribute2ItemFeature();

    result.widgets += self.navAttribute->select(attr |
attr.name.toLower().find("name") != 0 or attr.name.toLower().find("title") != 0)-
>NavAttribute2ItemFeature();

    result.widgets += object SLHyperlinkButton
    {
        name := "HyperlinkButton_SeeDetails_".getUniqueString();

        text := "See details";

        style := getDefaultTextStyle();

        height := 20;
        width := 150;
    }
}

mapping NavigationalClass::NavClass2Item(in container : SLStackPanel) :
SLStackPanel
{
    result.name := ("StackPanel_" + self.name + "_item").getUniqueString();

    result.navType := container.navType;
    result.isEnabled := true;
    result.isHidden := false;

    result.marginBottom := 2.0;

    result.style := 'Background="#ffaaaaaa"';

    var numElements := self.navAttribute->size();

    result.width := 500;
    result.height := numElements * 30 + 30;
    result.isHeightAuto := true;

    var setContentDataMethod := object WMethod
    {
        name := "SetContentData";
        type := PrimitiveType::Void;

        parameters += object WMethodParameter
        {
            name := "value";
            scope := ScopeKind::In;
            type := PrimitiveType::Object;
        };
    };

    result.methods += setContentDataMethod;

    var serviceLinks :=
inModel.rootObjects()[NavigationalModel].navigationalElem[ServiceLink]->select(l |
l.nodeOrigin = self);

    var modifyHlButton : SLHyperlinkButton;

    // HEADER
    result.widgets += object SLStackPanel
    {
        name := ("StackPanel_" + self.name + "_itemTitle").getUniqueString();

        horizontalAlign := true;

        height := 30;
        width := 500;

        widgets += object SLTextBlock
        {
            name := ("TextBlock_" + self.name + "_attrValue").getUniqueString();
            width := 250;
            height := 30;
            textWrap := true;
            text := self.name;
            textAlignment := HorizontalAlignment::Center;

            style := getDefaultTextStyle();
        };

        // DELETE FORM
        widgets += serviceLinks->select(l |
l.navOperation.referToOperation.operationType = OperationType::Destroy)-
>asSequence()->first().map ServiceLink2DestroyForm(container.navType, container);
    };

    // ITEM FEATURES
    result.widgets += self.navAttribute->NavAttribute2ItemFeature();

    // MODIFY FORMS
    var modifiers := serviceLinks->select(l |
l.navOperation.referToOperation.operationType = OperationType::Modifier);

    if (modifiers->size() <> 0)
    then
    {
        result.height := result.height * 2 - 30;

        var modifyForm := modifiers->asSequence()->first().map
ServiceLink2ModifyForm(container.navType, container);
        modifyForm.width := 482;

        var expander : SLExpander := addExpander(modifyForm);
        expander.header := "Modify " + container.navType.referToClass.name;

        result.widgets += expander;
    }
    endif;

}

helper NavigationalAttribute::NavAttribute2StackPanel() : SLStackPanel {}

helper NavigationalAttribute::NavAttribute2ItemFeature() : SLStackPanel {}

mapping TravesalLink::TraversalLink2ItemList () : SLStackPanel
{
    /* Generate the list of elements */
}

abstract mapping ServiceLink::ServiceLink2Form(in context : NavigationalClass, in
updatedList : SLStackPanel) : SLStackPanel
{
    result.name := ("StackPanel_" + self.name + "_form").getUniqueString();

    result.isEnabled := true;
    result.isHidden := false;
}

mapping ServiceLink::ServiceLink2NewForm(in navigationalClass : NavigationalClass,
in updatedList : SLStackPanel) : SLStackPanel
inherits ServiceLink::ServiceLink2Form
when {self.navOperation.referToOperation.operationType = OperationType::New}
{
    result.style := 'Background="#ffaaaaaa"';

    var numElements := self.argumentLink->size();

    result.width := 500;

    result.height := numElements * 30 + 30;

    result.widgets += self.argumentLink->map ArgumentLink2FormElement();

    result.widgets += self.map ServiceLink2FormButton(result, updatedList);
}

mapping ServiceLink::ServiceLink2FormButton(in father : SLStackPanel, in
updatedList : SLStackPanel) : SLButton
when {self.navOperation.referToOperation.operationType = OperationType::New}
{
    result.name := ("Button_" + self.name + "_form").getUniqueString();

    result.text := "New";

    result.horizontalContentAlignment := HorizontalAlignment::Center;
    result.verticalContentAlignment := VerticalAlignment::Center;

    result.type := TypeButton::SimpleButton;

    result.width := 100;
    result.height := 30;

    var newEvent := object WEvent { name := "OnClick" };
    result.events += newEvent;

    result.ecas += object EventCall
    {
        event := newEvent;

        conditions += object Condition
        {
            expresion := "true";

            trueActions += object ActionCall
            {
                action := object ServerAction
                {
                    navigationalAssociation := self;

                    // Update list containing the elements
                    if (updatedList != null)
                    then
                    {
                        onSuccessConditions += object Condition
                        {
                            expresion := "true";

                            trueActions += updatedList.map List2EcaUpdateList();
                        }
                    }
                    endif;
                };

                var formElements := father.widgets[SLStackPanel].widgets;

                self.argumentLink->forEach(argLink)
                {
                    arguments += object ServerArgument
                    {
                        argumentLink := argLink;

                        binding := object ActionArgumentViewBinding
                        {
                            _property := formElements->select(w |
w.name.find(argLink.name + "_argValue") <> 0)
                                        ->first().properties->first();
                        };
                    };
                };
            };
        };
    };
}

mapping ArgumentLink::ArgumentLink2FormElement() : SLStackPanel
{
    result.name := ("StackPanel_" + self.name + "_arg").getUniqueString();

    result.isEnabled := true;
    result.isHidden := false;

    result.width := 400;
    result.height := 30;

    result.isWidthAuto := true;
    result.isHeightAuto := true;

    result.horizontalAlign := true;

    result.widgets += object SLTextBlock
    {
        name := ("TextBlock_" + self.name + "_argName").getUniqueString();
        width := 50;
        height := 30;
        text := self.name;
    };

    var newWidget : SLWidget;

    switch
    {
        case (self.argument.collectionType != CollectionType::None)
        {
            if (self.argument.type = PrimitiveType::OID)
            then
            {
                newWidget := object SLDataGrid
                {
                    name := ("DataGrid_" + self.name +
"_argValue").getUniqueString();

                    width := 250;
                    height := 250;

                    //navigation :=
inModel.rootObjects()[NavigationalModel].navigationalElem[TravesalLink]->select(t |
t.nodeOrigin.name = "Home" and
t.nodeTarget.oclAsType(NavigationalClass).referToClass = self.argument.typeObject)-
>asSequence()->first();
                    //navType :=
navigation.oclAsType(TravesalLink).nodeTarget.oclAsType(NavigationalClass);

                    properties += object WidgetProperty
                    {
                        name := "selectedOIDs";
                        type := "OID";
                        //collectionType := CollectionType::Hash; // Mal pero no
funciona CollectionType::List
                    }
                }
            }
            else
            {
                newWidget := object SLListBox
                {
                    name := ("ListBox_" + self.name +
"_argValue").getUniqueString();

                    width := 150;
                    isHeightAuto := true;

                    style := getDefaultTextStyle();
                }
            }
            endif;
        }
        case (self.argument.type = PrimitiveType::Password)
        {
            newWidget := object SLPasswordBox
            {
                name := ("PasswordBox_" + self.name +
"_argValue").getUniqueString();
                width := 150;
                type := TypeTextBox::TextPassword;

                style := getDefaultTextStyle();

                properties += object WidgetProperty
                {
                    name := "password";
                    type := "String";
                };
            };
        }
        case (self.argument.type = PrimitiveType::Date)
        {
            newWidget := object SLDatePicker
            {
                name := ("DatePicker_" + self.name +
"_argValue").getUniqueString();
                width := 150;

                displayDate := "";
                selectedDate := "";
                selectedDateFormat := 0;

                style := getDefaultTextStyle();

                properties += object WidgetProperty
                {
                    name := "selectedDate";
                    type := "Date";
                };
            }
        }
        case (self.argument.type = PrimitiveType::Boolean)
        {
            newWidget := object SLCheckBox
            {
                name := ("CheckBox_" + self.name + "_argValue").getUniqueString();
                width := 150;

                text := "";
                type := TypeButton::CheckButton;

                style := getDefaultTextStyle();

                properties += object WidgetProperty
                {
                    name := "checked";
                    type := "Boolean";
                }
            };
        }
        case (self.argument.type = PrimitiveType::Enum)
        {
            newWidget := object SLComboBox
            {
                name := ("ComboBox_" + self.name + "_argValue").getUniqueString();

                width := 150;

                style := getDefaultTextStyle();

                var newOptions : String := "";

                self.argument.typeEnum.enumerationLiterals->forEach(lit)
                {
                    newOptions := newOptions + lit.name + "\n";
                };

                options := newOptions;

                properties += object WidgetProperty
                {
                    name := "selectedValue";
                    type := "Object";
                };
            };
        }
        else
        {
            newWidget := object SLTextBox
            {
                name := ("TextBox_" + self.name + "_argValue").getUniqueString();
                width := 350;

                style := getDefaultTextStyle();

                properties += object WidgetProperty
                {
                    name := "text";
                    type := "String";
                };
            };
        }
    };

    newWidget.height := 30;
    newWidget.isEnabled := true;
    newWidget.isHidden := false;

    result.widgets += newWidget;
}

mapping SLStackPanel::List2EcaUpdateList() : ActionCall
{
    result.name := "";

    var action = self.methods->selectOne(m | m.name.find("RefreshContentData") <>
0);
    if (action != null)
    then
    {
        result.action := object ClientAction { wMethod := action };
    }
    endif;
}

mapping ServiceLink::ServiceLink2DestroyForm(in context : NavigationalClass, in
updatedList : SLStackPanel) : SLStackPanel
inherits ServiceLink::ServiceLink2Form
when {self.navOperation.referToOperation.operationType = OperationType::Destroy}
{
    /* Create destroy form */
}

mapping ServiceLink::ServiceLink2ModifyForm(in context : NavigationalClass, in
updatedList : SLStackPanel) : SLStackPanel
inherits ServiceLink::ServiceLink2Form
when {self.navOperation.referToOperation.operationType = OperationType::Modifier}
{
    /* Create modify form */
}

/* Helpers */

mapping inout SLStackPanel::StackPanel2ScrollViewer() : SLScrollViewer {}

helper addScrollViewer(inout sp : SLStackPanel) : SLScrollViewer {}

helper addExpander(inout sp : SLWidget) : SLExpander {}

helper String::getUniqueString() : String {}

helper getDefaultTextStyle() : String {}
```

Annex G. IMPLEMENTATION DETAILS

This annex describes all the aspects related to the processes of generation and the implementation of *S^m4RIA extension for OIDE* that were not detailed in Chapter 7 and Chapter 8.

G.1. STRUCTURE OF THE PROJECT GENERATED

The following table details the structure of the Visual Studio projects proposed in order to contain the code generated.

Table G.7 Structure of the Visual Studio solution generated and the description of the elements.

Project	Project Element
`InitializeDB/`	
	`CreateDB.cs`
	`InitializeDB.csproj`
	`Program.cs`
	`Properties/`
	`app.config`
`SilverlightCommon`	
`<SolutionName>NHibernate`	
	`AppLib/`
	`DAC/`
	`BEC/`
	`DTO/`
	`Exceptions/`
	`Mappings/`
	`NHibernateHelper.cs`
	`Properties/`
	`Resources/`
	`Utils/`
	`app.config`
	`hibernate.cfg.xml`
	`<SolutionName>NHibernate.csproj`
`<SolutionName>_LinkedDataCommon`	
	`App.config`
	`Assemblies/`
	`Clients/`
	`Entities/`
	`Gateways/`
	`Ontologies/`

Project	Project Element
	`Properties/`
	`<SolutionName>_LinkedDataCommon.csproj`
`<SolutionName>_WCF`	
	`Assemblers/`
	`AssemblersEN/`
	`DTO/`
	`Properties/`
	`Service.svc / Service.svc.cs / ServiceExternal.svc.cs`
	`Web.config`
	`<SolutionName>_WCF.csproj`
	`clientaccesspolicy.xml`
`<SolutionName>Silverlight`	
	`App.xaml / App.xaml.cs`
	`MainPage.xaml / MainPage.xaml.cs`
	`Properties/`
	`ResourceDictionary.xaml`
	`Resources/`
	`Service/`
	`ServiceReferences.ClientConfig`
	`<SolutionName>Silverlight.csproj`
	`UIEntities/`
	`ViewModels/`
	`Views/`
`<SolutionName>Silverlight.Web`	
	`Properties`
	`Silverlight.js`
	`<SolutionName>Silverlight.Web.csproj`
	`Web.config`
	`index.aspx`
	`index.html`

G.2. XPAND REFERENCE

This section briefly describes the elements of the Xpand language for the definition of model-to-text transformation rules. For a more complete explanation please check the following Web site:

`http://www.openarchitectureware.org/pub/documentation/4.3.1/html/contents/core_reference.html`

Table G.8 shows the main elements of the language and their description.

Table G.8. Description of the main elements of the Xpand language.

Xpand Element	Description

`«DEFINE rule FOR metaclass»`
`«ENDDEFINE»`
	Definition of a transformation rule for the classes of a given metaclass.

`«EXPAND rule FOREACH collection»`
`«EXPAND rule FOR object»`
	Invocation of a transformation rule previously defined.

`«PROTECT CSTART "opening" CEND "closing" ID "idString"»`
`«ENDPROTECT»`
	Definition of a protected region, in which the changes in the generated code are preserved among generation processes.

`«GLOBALVAR varName»`	Use of a global variable. Global variables are defined in the workflows that invoke the transformation rules from the Eclipse framework.

`«IF condition»`
`«ELSEIF condition»`
`«ELSE»`
`«ENDIF»`
	If-else block

`«LET expression AS variable»`
`«ENDLET»`
	Definition of a local variable.

`«FOREACH collection AS object»`
`«ENDFOREACH»`
	Foreach loop.

`«REM»Comment«ENDREM»`	Comment.

G.3. XTEXT GRAMMAR OF THE EXTENDED DOMAIN MODEL EDITOR

This section contains the grammar rules used for the development of the textual editor for the Extended Domain Model (see Table G.9). For more details about the language for grammar defition, please check the following Web site:

http://www.eclipse.org/Xtext/documentation.html#grammarLanguage

Table G.9. Grammar rules of the Xtext editor for the Extended Domain Model.

```
grammar es.ua.dlsi.ooh.sm4ria.extendedDomainModel.xtext.EdmText with
org.eclipse.xtext.common.Terminals

import
"platform:/resource/es.ua.dlsi.ooh.sm4ria.extendedDomainModel/model/ExtendedDomainM
odel.ecore"

import "http://www.eclipse.org/emf/2002/Ecore" as ecore
```

```
import
"platform:/resource/net.insidesoft.conceptualView/model/ConceptualView.ecore" as
conceptualView

EDModel returns EDModel:
        {EDModel}
        (imports+=Import)*
        'create'
        name=EString ';'

OntologyModel returns OntologyModel:
        OntologyModel_Impl | Source;

ModelRelation returns ModelRelation:
        OntoImport | Instance;

OntologyElement returns OntologyElement:
        Concept | Attribute_Impl | ObjectProperty | /*Individual |*/ Inheritance |
Association_Impl | RefAttribute | LinkedAssociation;

Property returns Property:
        Attribute_Impl | ObjectProperty | RefAttribute;

OntologyModel_Impl returns OntologyModel:
        {OntologyModel}
        (isLocal?='local')?
        'OntologyModel'
        name=EString
        ('<'
        uriBase=EString ('as' namespace=EString)?
        '>')?
        ('refersTo'
        conceptualModel=[conceptualView::ConceptualModel|Fqn])?
        (('{'
                ('desc' '=' description=EString)?
                ( elements+=OntologyElement )*
        '}') | ';');

Import returns Import:
        'imports' importedNamespace=FqnWithWildCard;

Instance returns Instance:
        {Instance}
        'Instances' id=EString ':' base=[OntologyModel|Fqn] '->'
target=[Source|Fqn] ';'
        ;

OntoImport returns OntoImport:
        {OntoImport}
        'OntoImport' id=EString ':' base=[OntologyModel|Fqn] '->'
target=[OntologyModel|Fqn] ';'
        ;

Source returns Source:
        {Source}
        (isLocal?='local')?
        (type=SourceType)
        'Source'
        name=EString
        ('<' uriBase=EString '>')?
        (('{'
                ('desc' description=EString)?
        '}') | ';');

Inheritance returns Inheritance:
        {Inheritance}
        (visibility=Visibility)?
        'Generalisation'
        name=EString
        '('
        descendant=[OntologyElement|Fqn] '=>' ascendant=[OntologyElement|Fqn]
        ')'
        ('refersTo' conceptualInheritance=[conceptualView::Inheritance|Fqn])?
        (('{'
                ('desc' '=' description=EString)?
        '}') | ';');

Concept returns Concept:
        {Concept}
        (visibility=Visibility)?
        'Concept'
        name=EString
        ('<' uri=EString '>')?
        ('refersTo' domainClass=[conceptualView::Class|Fqn])?
        (('{'
                ('desc' '=' description=EString)?
                (properties+=Property)*
        '}') | ';');

Attribute returns Attribute:
        Attribute_Impl | RefAttribute;

Attribute_Impl returns Attribute:
        {Attribute}
        (visibility=Visibility)?
        target=XmlDatatypes
        'Attribute'
        name=EString
        ('<' uri=EString '>')?
        ('min' minCardinality=EInt)?
        ('max' maxCardinality=EInt)?
        ('refersTo' domainAttribute=[conceptualView::Attribute|Fqn])?
        (('{'
                ('desc' '=' description=EString)?
        '}') | ';');

RefAttribute returns RefAttribute:
        (visibility=Visibility)?
        target=XmlDatatypes
        'RefAttribute'
        name=EString
        ('<' uri=EString '>')?
        ('min' minCardinality=EInt)?
        ('max' maxCardinality=EInt)?
        ('refersTo' domainAttribute=[conceptualView::Attribute|Fqn])?
        '->' refProperty=[Property|Fqn]
        (('{'
                ('desc' '=' description=EString)?
        '}') | ';');

ObjectProperty returns ObjectProperty:
        {ObjectProperty}
        (visibility=Visibility)?
        (isSymmetric?='symmetric')?
        (isTransitive?='transitive')?
        'ObjectProperty'
        name=EString
        ('<' uri=EString '>')?
        ('min' minCardinality=EInt)?
        ('max' maxCardinality=EInt)?
        ('refersTo' domainAttribute=[conceptualView::Attribute|Fqn])?
        (('{'
                ('desc' '=' description=EString)?
                ('joinText' '=' joinText=EString)?
        '}') | ';');

Individual returns Individual:
        {Individual}
        'Individual'
        name=EString
        '{'
                ('uri' uri=EString)?
                ('description' description=EString)?
                ('visibility' visibility=Visibility)?
        '}';

Association_Impl returns Association:
        (visibility=Visibility)?
        'Association'
        name=EString
        ('<' uri=EString '>')?
        '(' conceptOrigin=[Concept|Fqn] '=>' conceptTarget=[Concept|Fqn] ','
direct=[ObjectProperty|Fqn] ('<->' inverse=[ObjectProperty|Fqn])? ')'
        (('{'
                ('desc' '=' description=EString)?
        '}') | ';');

LinkedAssociation returns LinkedAssociation:
        (visibility=Visibility)?
        'LinkedAssociation'
        name=EString
        ('<' uri=EString '>')?
        '(' conceptOrigin=[Concept|Fqn] '=>' conceptTarget=[Concept|Fqn] ','
direct=[ObjectProperty|Fqn] ('<->' inverse=[ObjectProperty|Fqn])? ')'
        'refersTo' domainAssociation=[conceptualView::Association|Fqn]
        (('{'
                ('desc' '=' description=EString)?
        '}') | ';');


enum Visibility returns Visibility:
        Public = 'public' | Private = 'private';

enum XmlDatatypes returns XmlDatatypes:
        string = 'String' | boolean = 'Boolean' | decimal = 'Decimal' | float =
'Float' | double = 'Double' | duration = 'Duration' | dateTime = 'DateTime' | time
= 'Time' | date = 'Date' | gYearMonth = 'GYearMonth' | gYear = 'GYear' | gMonthDay
= 'GMonthDay' | gDay = 'GDay' | gMonth = 'GMonth' | hexBinary = 'HexBinary' |
base64Binary = 'Base64Binary' | anyURI = 'AnyURI' | QName = 'QName' | NOTATION =
'NOTATION' | integer = 'Integer';

enum SourceType returns SourceType:
        SPARQL = 'SPARQL' | SWS = 'SWS';

FqnWithWildCard returns ecore::EString:
        Fqn ( '.*' )?;

Fqn returns ecore::EString:
        EString ( '.' EString )*;

EString returns ecore::EString:
        STRING | ID;

EBoolean returns ecore::EBoolean:
        'true' | 'false';

EInt returns ecore::EInt:
        '-'? INT;
```

Annex H. Resumen en español

Este anexo presenta un resumen en español de los principales problemas que trata de solucionar esta tesis y de las contribuciones realizadas con ese fin.

H.1. Introducción

En las sociedades modernas las necesidades de información están creciendo de forma exponencial. Durante las dos últimas decadas, Internet ha experimentado una contínua, relativamente rápida, evolución desde diferentes puntos de vista, cuyos objetivos se superponen en algunos casos, orientados a satisfacer las necesidades de información de los usuarios. Este hecho ha llevado a la creación de diferentes tendencias centradas en satisfacer un subconjunto de los requerimientos de usuario. Murugesan (Murugesan, 2008) indentificó algunos de ellas: Web 1.0, Web 2.0, *Rich Internet Applications*, la Web Semántica y la Web móvil (aunque podrían existir otras).

De estas tendencias identificadas, esta tesis se centra en las *Rich Internet Applications* (en español, aplicaciones enriquecidas de Internet), es decir, aplicaciones creadas en la Web 2.0 en la misma época que las aplicaciones sociales, cuya interfaz de usuario proveen funcionalidades hasta aquel momento sólo vistas en interfaces de usuario de escritorio. Basadas en tecnologías como Flex, Silverlight o jQuery (entre otras), estas aplicaciones incluyen interfaces de usuario con un alto nivel de interactividad y dinamicidad, que incluyen elementos multimedia y pueden recuperar datos del servidor Web sin cambiar de página Web o presionar un enlace o un botón.

Una cuestión de acceder y compartir datos: interoperabilidad de datos en Rich Internet Applications.

Sin embargo, los buscadores Web actuales, los cuales son para muchos usuarios el punto de entrada a la información de la Web, no pueden acceder e indexar la información contenida en las *Rich Internet*

Applications. Como consecuencia, los usuarios no pueden encontrar fácilmente el contenido que muestran. Este hecho puede hacer que los desarrolladores software y las empresas eviten su uso a pesar de los beneficios que tienen para la visualización de los datos.

El comportamiento de las interfaces RIA esta dirigido por los eventos de usuario, es decir, muestran la información en base a las demandas de los usuarios, expresadas por medio de ciertos eventos en los componentes de la interfaz. Este proceso complica el acceso a los datos a los agentes software independientemente de la tecnología que se haya utilizado para desarrollar las aplicaciones. En este escenario, las RIAs implementadas con HTML tienen una ventaja sobre las RIA implemenetadas tecnologías orientadas a componentes (o *plug-in*), p.e., Silverlight o Flex, ya que los contenidos son visualizados por medio de representaciones textuales en el código HTML, de forma similar a las interfaces HTML tradicionales de la Web 1.0.

Reinvirtiendo los esfuerzos en la Web.

Dada la madurez de Internet y de las diferentes tecnologías desarrolladas bajo su paraguas, la solución a los problemas encontrados en las RIA podría obtenerse a partir de los esfuerzos ya realizados en otras áreas de la Web. En este caso, más concretamente, pueden usarse las técnicas y tecnologías para la gestión de conocimiento desarrolladas en la Web Semántica. La Web Semántica (Berners-Lee et al., 2001) considera los sistemas software como usuarios de primera clase que ayudan a los usuarios humanos en sus tareas; no únicamente meras herramientas para la gestion y visualización de la información, sino que también pueden realizar tareas de adquisición y gestión de conocimiento y participar en la toma de decisiones. Siguiendo este objetivo, nuevas tecnologías y herramientas han sido desarrolladas para proveer de un significado explícito y desambiguado a la información que recorre la Web usando técnicas para la captura, representación y gestión de conocimiento. Entender el significado del contenido de los sitios Web mejora la interoperabilidad (a tres niveles: léxico, sintáctico y semántico) de los componentes software de la Web, característica que carecen las *Rich Internet Applications*.

Retos de desarrollo y la ingeniería dirigida por modelos.

Para desarrollar aplicaciones que combinen características de las diferentes tendencias existentes en la Web es necesario balancear diferentes factores y asumir ciertos retos. Como Murugesan indicó (Murugesan, 2008), todos estos se pueden resumir en un único reto: "*diseñar y desarrollar sistemas Web para una mejor a) usabilidad, diseño de interfaz y navegación; b) comprensión; c) rendimiento; d) seguridad e integridad; e) evolución, crecimiento y mantenibilidad; f) testeo; y g) movilidad*".

El éxito de la solución propuesta para RIA y su aceptación dependerán de forma directa en el coste de asumir estos retos, el cual es normalmente alto en términos de recursos y tiempo debido a la complejidad de las funcionalidades necesarias por las aplicaciones y la dinamicidad del escenario Web. En esta última década, diversas metodologías dirigidas por modelos para el desarrollo de aplicaciones Web han tratado los retos mencionados facilitando los procesos de creación de aplicaciones Web complejas. Estas metodologías proponen procesos de desarrollo en los cuales las actividades están orientadas al diseño de modelos software. Asimismo, definen una colección de transformaciones para obtener nuevos modelos a partir de modelos existentes o directamente los componentes software de la aplicación diseñada. Este tipo de técnicas de desarrollo, junto con una herramienta software que las implenente y soporte, pueden reducir los costes asociados con el desarrollo de aplicaciones Web complejas.

No obstante, ninguna de las metodologias actuales combina de forma efectiva los elementos necesarios para el desarrollo de una solucion para los problemas detectados en RIA. Las metodologías actuales (p.e., WebML) contienen parte de los elementos necesarios (p.e., desarrollo de interfaces ricas, ontologías o acceso a servicios Web) pero inconexos. Además, las soluciones no están alineadas completamente hacia nuevas iniciativas para gestionar y compartir conocimiento en la Web Semántica, como la Web de Datos.

La investigación realizada en esta tesis intenta contestar a las siguientes preguntas de investigación, planteadas a partir de los problemas detectados en el escenario descrito:

RQ1 *– ¿Es posible mejorar la interoperabilidad de las Rich Internet Applications con otros sistemas software (como, por ejemplo, los buscadores Web) usando técnicas, tecnologías y recursos de la Web Semántica?*

RQ2 *– ¿Cómo pueden las actuales metodologías dirigidas por modelos ser extendidas para desarrollar la solución propuesta a los problemas detectados en las Rich Internet Applications?*

RQ3 *– ¿Cómo se pueden implementar las soluciones propuestas en una herramienta software CASE?*

Para cada una de las cuestiones planteadas, se define un conjunto de objetivos que deben de ser cumplidos para poder contestar a las preguntas. Los objetivos propuestos para cada pregunta son los siguientes:

Objetivo 1) *Mejorar la interoperabilidad de las Rich Internet Applications con los sistemas de la Web que usan el texto como entrada (e.g., buscadores o lectores para invidentes).*

O1.1) Mejorar la exportabilidad de los datos contenidos en Rich Internet Applications.

O1.2) Mejorar el acceso a la información relativa a los elementos multimedia.

O1.3) Combinar técnicas, tecnologías y recursos ya existentes en la Web Semantica con tecnologías para la creación de Rich Internet Applications.

O1.4) Desarrollar una colección de casos de estudio para evaluar la validez de la solución propuesta.

Objetivo *2) Diseñar una metodología de desarrollo software dirigido por modelos para el desarrollo de la solución propuesta.*

O2.1) Facilitar el desarrollo de la solución propuesta en O1.

O2.2) Mejorar la mantenibilidad de la solución propuesta en O1.

O2.3) Extender una metodología existente para el desarrollo de RIA.

Objetivo *3) Desarrollar una herramienta software CASE que soporte los elementos de la metodología diseñada.*

Los siguientes apartados introducen las principales contribuciones de la tesis, desarrolladas para cumplir con los objetivos propuestos.

H.2. Rich Internet Applications en la Web Semántica

En la última decada, diversos autores (Meliá et al. 2008; Linaje et al. 2007; Fraternali, Comai, et al. 2010) han tratado de especificar un conjunto de requisitos deseables para cualquier RIA y la forma en que deberían ser desarrollados utilizando técnicas de desarrollo dirigidos por modelos. No obstante, la combinación de técnicas de la Web Semántica con metodologías para el desarrollo de RIA no ha sido estudiada en profundidad. Dado que las tecnologías de la Web Semántica están especializadas en representar y compartir conocimientos, la alianza entre las dos aproximaciones puede solventar los problemas encontrados en las RIA independientemente de la tecnología RIA empleada, es decir, que podría ser aplicada en cualquier tipo de RIA.

En este apartado se presenta la primera contribución de esta tesis: el concepto de *Semantic Rich Internet Application (SRIA)*, que define a un nuevo tipo de RIA que usa de forma extensiva las técnicas, tecnologías y recursos de la Web Semántica para compartir sus propios datos y reusa datos de otras fuentes para enriquecer su propio contenido.

H.2.1. Requisitos

La definición de un conjunto concreto de requisitos de una aplicación Web facilita la identificación de sus principales metas y componentes software. En este caso, los requisitos también permiten apreciar de forma más sencilla las diferencias entre las RIA tradicionales y este nuevo tipo. Los requisitos para el desarrollo de SRIA combinan aspectos de RIA y otros aspectos relacionados con las aplicaciones Web semánticas de tal forma que las aplicaciones resultantes pueden ser consideradas como una combinación de ambas.

Este apartado propone una lista específica de requisitos para caracterizar a las SRIAs, centrándose en aquellos requisitos que no considerados en el desarrollo de RIA tradicionales. Esta lista toma en consideracion los estudios realizados por otros autores, tales como Brambilla y Facca (Brambilla and Facca, 2007) y Roval et al. (Rovan et al., 2011), centrados en el desarrollo de aplicaciones de la Web Semántica, así como la arquitectura de la Web Semántica y los principios de *Linked Data*, descritos en el apartado 2.1.2 (página 19).

Las características propuestas pueden resumirse en dos requisitos no funcionales de alto nivel. El cumplimiento de estos primeros requisitos está asociado al cumplimiento de una serie de requisitos funcionales, que restringen las funcionalidades de las aplicaciones resultantes. La lista de requisitos propuesta es la siguiente:

R1) Alto nivel de exportabilidad y reusabilidad del contenido de la aplicación. La aplicación tiene que ser capaz de proveer sus contenidos de una forma desambiguada y estructurada a los agentes software o incluso a otras RIA semánticas.

Rf1.1) La aplicación tiene que usar ontologías como formalismo para la representación de conocimiento. Todos los datos almacenados y gestionados por una SRIA tienen que ser representados por medio de ontologías, que son el estándar para la representación de conocimiento en la Web Semántica.

Rf1.2) La aplicación tiene que proveer anotaciones semánticas del contenido. Las ontologías proveen un método para representar y estructurar el conocimiendo utilizado por una SRIA. Sin embargo, es también necesario mapear los datos de la aplicación en instancias de la ontología y anotar algunos fragmentos de información para representar de forma efectiva que información se esta mostrando en un momento determinado. Esta información se define en el modelo de anotación (Bettencourt et al., 2006) propuesto para SRIAs.

R2) Alto nivel de reusabilidad del conocimiento externo a la aplicación. Siguiendo la filosofía de la Web Semántica y los principios de *Linked Data* en la Web de Datos, los contenidos de la aplicación deben de ser enriquecidos con conocimiento de otras fuentes. La aplicación no tiene que ser aislada sino capaz de obtener conocimiento de diferentes fuentes de la Web Semántica. Este requerimiento puede alcanzarse por medio de los siguientes subrequerimientos:

Rf2.1) La aplicación tiene que reusar ontologías existentes. Como resultado, será posible interconectar conocimiento entre una red de aplicaciones. Además puede simplificar los procesos de compartir conocimiento (R1) y los procesos de desarrollo de aplicaciones similares a partir de una aplicación.

Rf2.2) La aplicación tiene que reusar bases de conocimiento existentes. Con las instancias obtenidas desde otras fuentes en la Web, sería posible enriquecer los contenidos que se muestran a los usuarios por medio de agregaciones de datos. De forma inicial, para limitar la complejidad de la solución, solo dos tipos de fuentes de conocimiento son tomadas en consideración:

Rf2.2.1) La aplicación tiene que reusar conocimiento disponible en la Web de Datos como Linked Data. La aplicación utilizará instancias de ontología de los conjuntos de datos de la Web de Datos, que se encuentran repartidos por toda la Web.

Rf2.2.2) La aplicación puede reusar conocimiento desde otras aplicaciones. El contenido de las SRIA debería ser compatible entre sí, es decir, el conocimiento compartido por una aplicación debería poder ser consumido por otras aplicaciones siguiendo los principios de *Linked Data*.

En esta lista de requisitos, ontologías y bases de conocimiento, que contienen las instancias de ontología, son considerados elementos diferentes, a pesar de que en diversos trabajos, p.e., Gomez-Perez et al. (Gómez-Pérez et al., 2007), las instancias de una ontología son tratadas como parte de la misma. Aunque puede parecer una decisión controvertida, esta fue tomada en base a la definición de ontología de Gruber (Gruber, 1995), que respalda Guarino (Guarino, 1998): "*una ontología sirve a un proposito diferente que un estado de una base de conocimiento*". Mientras que las instancias contenidas en las ontologías pueden ser consideradas como conocimiento compartido en el dominio, las instancias contenidas en una base de conocimiento "*pueden incluir el conocimiento necesario para solucionar un problema o contestar preguntas arbitrarias sobre un dominio*". Esta aproximación facilita la conceptualización de aquellos repositorios que, con diferentes objetivos, almacenan instancias de una misma ontología, lo cual no es inusual en la Web Semántica o en la Web de Datos.

H.2.2. Estructura

Una vez clasificados los requisitos de la aplicación, el siguiente paso es la definición de una estructura de la aplicación, que represente los principales modulos software de la SRIA y sus funcionalidades. La Figure H.17 representa un esquema con la propuesta de aplicación SRIA incluyendo las asocicaciones con otros componentes de la Web.

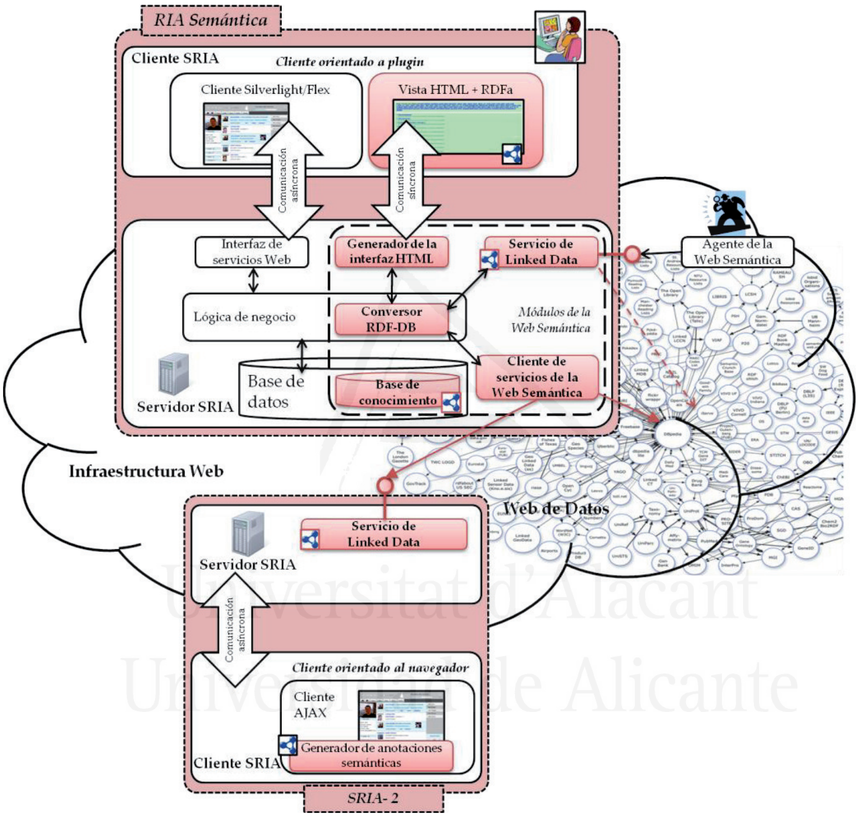


Figure H.17. Esquema de la estructura de una Semantic Rich Internet Application.

De forma similar a las RIA, las SRIA se desarrollan utilizando una arquitectura cliente-servidor cuyos clientes, que contienen interfaces de usuario ricas, invocan los servicios ofertados por los servidores por medio de procesos de comunicacion asíncronos. De esta forma, los clientes y los servidores pueden ser totalmente desacoplados.

El servidor SRIA reusa parte de los componentes originales de la RIA, más específicamente, aquellos componentes que realizan las operaciones básicas sobre datos:

1) **Base de datos**, que gestiona el almacenamiento persistente de los datos.
2) **Lógica de negocio**, que incluye todos los componentes que realizan las principales tareas de la aplicación y gestionan los datos recuperados de la base de datos.
3) **Interfaz de servicios Web**, que ofrece un conjunto de servicios desde el servidor a la interfaz que usuario. Dichos servicios proporcionan acceso a los datos del servidor y a las funciones de la lógica de negocio.

En cuanto a los clientes SRIA, sus interfaces reutilizan la mayor parte de los componentes de las interfaces de usuario de las RIA tradicionales. En base a la tecnológica de implementación, el cliente (S)RIA puede ser clasificado en dos categorías: orientado a plugin (Figure H.17, SRIA) o orientado al navegador (SRIA-2), de la misma forma que los clientes RIA tradicionales. La clasificación de los clientes RIA se explicó en el apartado 2.1.1 (página 14).

Como muestra la figura, a parte de los módulos heredados de las RIA, las SRIA incluyen un conjunto de nuevos módulos software para satisfacer los requerimientos propuestos. Estos módulos realizan funciones relacionadas con la reutilización de conomiento. Los módulos pueden ser descritos de la siguiente forma:

4) **Base de conocimiento (módulo de servidor).** Este módulo gestiona la base de conocimiento de la aplicacion, la cual almacena las instancias de la ontologia usada por la aplicacion (basada en RDF, *Resource Description Framework*). Dada la necesidad de reutilizar conocimiento de la Web Semántica, las SRIA necesitan una base de conocimiento, que puede ser implementada sobre la base de datos existente.
5) **Servicio de Linked data (módulo de servidor).** Este módulo ofrece un servicio para acceder a parte del conocimiento almacenado en la base de conocimiento de la SRIA. En este caso, esta aproximación esta alineada con los principios de *Linked Data* y el protocolo SPARQL para RDF (World Wide Web Consortium, 2008b). No obstante, en función de los requisitos de la aplicación, esta interfaz podría ser cambiada por otra, por ejemplo, basada en servicios Web semánticos. Dado que la estructura de la consulta puede afectar de forma directa el rendimiento del

servicio, en esta propuesta, el servicio SPARQL puede limitar el acceso a un determinado número de clases o instancias de una clase en base a las preferencias de los desarrolladores.

6) **Cliente de servicios de la Web Semántica (módulo de servidor).** Este cliente es en realidad una combinación de diferentes tipos de clientes: servicios Web (SOA, RES), servicios Web semánticos y servicios de Linked Data. Este modulo posibilita el acceso bajo demanda a otras ontologías y bases de conocimientos de la Web (incluso a otras SRIA).
7) **Generador de anotaciones semánticas (módulo de cliente, interfaces orientadas al navegador).** En SRIAs orientadas al navegador, el cliente puede incluir un módulo software que integre anotaciones RDFa, que enlacen el contenido mostrado en la interfaz de usuario con las instancias de ontología contenidas en la base de conocimiento o en fuentes de conocimiento externas. En el caso de SRIAs orientadas a plug-in, los usuarios podrían acceder a este conocimiento a través del servicio de Linked Data o the la vista HTML+RDFa, generada por el módulo siguiente.
8) **Generador de la interfaz HTML (módulo de servidor).** Este módulo genera una representación HTML de las instancias de ontologías contenidas en la base de conocimiento. Esta vista esta anotada utilizando código RDFa, que hace referencia a las instancias almacenadas en la base de conocimiento. A diferencia de las interfaces RIA, esta interfaz puede ser fácilmente procesada por los buscadores Web. El punto de acceso a esta interfaz es la URL incluida en la cabecera de la página Web HTML que contiene al cliente RIA. En esta intefaz, la comunicación entre el cliente y el servidor de la SRIA sigue un proceso síncrono, al igual que en las interfaces Web tradicionales.

H.2.3. Casos de estudio

Las RIA semánticas con una plataforma genérica para el desarrollo de diferentes aplicaciones que ofrece una solution independiente de la tecnologia a los principales problemas de las RIA. La evaluación de la propuesta fue realizada por medio del desarrollo de un conjunto de casos de estudio. El análisis cualitativo de las aplicaciones desarrolladas ayudo a mejorar la aproximación en un proceso iterativo. Cada uno de los casos de estudios fue validado de forma externas en conferencias nacionales e internacionales, así como en revistas internacionales. Este apartado introduce los cuatro casos de estudio desarrollados en el proceso de evaluación:

1) El desarrollo de un reproductor multimedia, inspirado por el caso de studio presentado por Brambilla y Facca (Brambilla and Facca, 2007). Este caso de estudio fue presentado inicialmente por Hermida et al. (Hermida et al., 2011b) y se describe en el Annex C.
2) El desarrollo de una red social en la Web Semántica, tal y como definen Kinsella et al. (Kinsella et al., 2009). Los detalles de esta aplicación se encuentran en el apartado 3.4.1. Este caso de estudio fue presentado inicialmente por Hermida et al. (Hermida et al., 2011a)
3) El desarrollo de una aplicación SRIA orientada a la Inteligencia de Negocio. La aproximación SRIA también fue aplicada en el campo de la Inteligencia de Negocio en un nuevo caso de estudio que consistió en el desarrollo de una aplicación para la gestión de empleados y procesos por medio de una red social. Este caso se describe en el apartado 3.5.2.

Los casos de estudio fueron desarrollados utilizando la plataforma .NET (C#) y, en especial, los frameworks *Windows Communication Foundation* (WCF, para los components de servidor) y *Silverlight* (para los componentes de cliente).

H.3. Una metodología para el desarrollo de Semantic Rich Internet Applications

Durante la última década, las metodologías de desarrollo Web dirigidas por modelos han probado su validez para llevar a cabo todas las fases del desarrollo de aplicaciones Web, incluso para RIA, facilitando el diseño y la generación sistematica de aplicaciones Web gracias a las herramientas CASE. Este tipo de metodologías puede ser una solución apropiada para el desarrollo de SRIA, dada la complejidad de su estructura, ya que pueden reducir el coste de desarrollo en términos de tiempo y recursos, minimizando de esta forma el riesgo de que el proyecto de desarrollo fracase. Estos factores son relevantes cuando se desarrollan aplicaciones en un entorno empresarial.

Estas metodologías son relativamente modernas y uno de los aspectos todavía no soportados es el desarrollo de RIAs capaces de gestionar información proveniente de la Web Semántica. Para facilitar el desarrollo de SRIAs, como segunda contribución, esta tesis presenta la metodología S^{m}4RIA, que se basa en la metodología OOH4RIA (Meliá et al. 2008), especializada en el desarrollo de RIAs tradicionales. El objetivo de esta metodología es cubrir todas las fases del desarrollo de las SRIA: desde el diseño de las entidades de datos y la interfaz de usuario hasta la generación de los módulos software.

Para cumplir los requisitos de las SRIA, la metodología S^{m}4RIA define nuevos procesos y artefactos, no incluidos en OOH4RIA:

(i) Dos nuevos metamodelos MOF, creados como una extensión del OMG *Ontology Definition Metamodel* (Object Modeling Group 2009), que definen dos nuevos modelos ontológicos;

(ii) Un conjunto de transformaciones modelo a modelo que pueden crear esbozos de los diferentes modelos S^{m}4RIA, los cuales pueden ayudar a los diseñadores en el proceso de creación de un modelo a partir otro;

(iii) Un conjunto de transformaciones modelo a texto que generan los nuevos módulos SRIA a partir de los modelos S^{m}4RIA.

Asimismo, la metodología S^{m}4RIA extiende el proceso de desarrollo de OOH4RIA:

(i) Incluyendo nuevos mecanismos de modelado a los modelos funcionales de OOH4RIA como una extensión del metamodelo MOF de OOH4RIA;

(ii) Adaptando las actividades que define OOH4RIA añadiendo nuevas tareas y modificando las existentes.

La Figure H.18 muestra una vista preliminar del proceso de desarrollo de S^{m}4RIA utilizando un diagrama de clases SPEM2, acorde a una extensión del metamodelo de SPEM2 que incluye aspectos no considerados en el metamodelo original, p.e., la representación de los motores de transformación (llamados *Model Transformers* en el modelo) utilizando el estereotipo *ProcessRole* y la representación de las transformaciones MDA utilizando una nueva colección de estereotipos que extienden la metaclase *TaskDefinition*, definida en SPEM, p.e., *PIM2PIM*, *PIM2PSM*, etc.

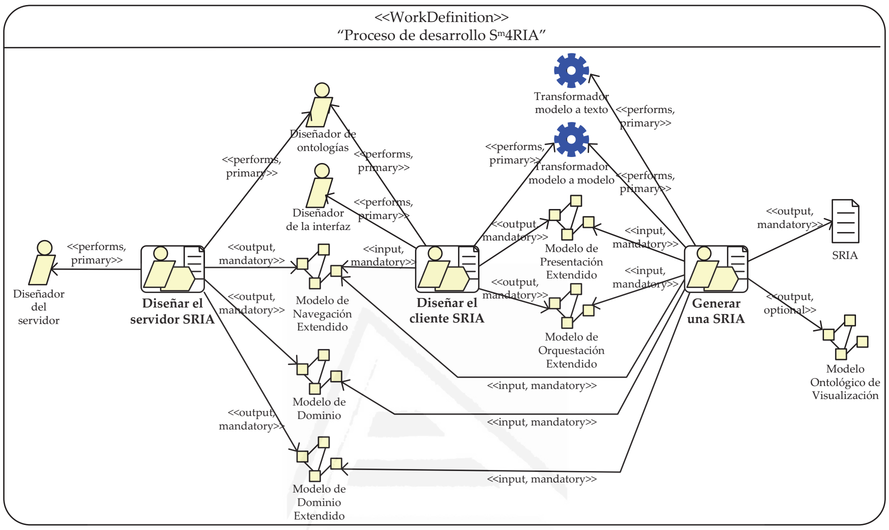


Figure H.18. Diagrama SPEM2 con el proceso de desarrollo Sm4RIA.

Los siguientes apartados explican cada uno de los componentes que aparecen en la figura: los roles de usuario, los modelos y el proceso de desarrollo. El resultado final del proceso, es decir, la SRIA, ha sido explicado en el apartado anterior.

H.3.1. LOS ROLES DE USUARIO EN S^M4RIA

Hay cinco tipos de rol de usuario involucrados en las diferentes actividades del proceso (ver Figure H.18):

a) **Diseñador del servidor.** El diseñador del servidor crea los componentes del servidor de la SRIA, p.e., la base de datos, los servicios web, etc.

b) **Diseñador de la interfaz de usuario.** El diseñador de la interfaz de usuario realiza aquellas tareas relacionadas con la construcción de la interfaz de usuario de la SRIA y la invocación de los servicios que provee el servidor SRIA.

c) **Diseñador de ontologías.** El diseñador de ontologías realiza las tareas relacionadas con la interconexión de la aplicación con fuentes de conocimiento externo. Estas tareas pueden ser realizadas también por el diseñador del servidor dependiendo de su perfil.

d) **Transformador de modelo a modelo.** Este rol corresponde al motor de transformaciones capaz de convertir un modelo en otro.

e) **Transformador de modelo a texto.** Este último rol corresponde al motor de transformaciones capaz de convertir el contenido de un modelo en código fuente de software.

H.3.2. Los modelos S^M^4RIA

Son seis los modelos involucrados en el proceso de desarrollo de una SRIA, que modelan diferentes cuestiones:

1. **Modelo de dominio** (*Domain Model*, modelo independiente de la plataforma). El modelo de Dominio, importado de OOH4RIA sin ningún cambio, define las principales estructuras de datos de la aplicación (en términos de clases y propiedades), las relaciones entre ellas y las operaciones que pueden realizarse. Las operaciones que se pueden definir se clasifican en dos grupos: operaciones CRUD (crear, leer, actualizar y borrar), que son las operaciones básicas sobre los datos, y las operaciones personalizadas o *custom*, que pueden ser definidas libremente por el diseñador. Asimismo, el modelo de Dominio permite definir mapeos entre los objetos de datos y las tuplas de la base de datos.
2. **Modelo extendido de dominio** (*Extended Domain Model*, EDM; modelo independiente de la plataforma). El EDM define ontologías ligeras que pueden representar las entidades del dominio de la aplicación y las relaciones entre ellas. Además, este modelo puede capturar las ontologías importadas desde otras fuentes y las bases de conocimiento disponibles para cada una de las ontologías modeladas. Los objetivos específicos de este modelo son los siguientes:
    a. Representar la ontología de dominio de la aplicación;
    b. Establecer relaciones entre la ontología de la SRIA y ontologías externas, alineando de esta forma los elementos

del dominio con elementos externos (o incluso reutilizando elementos externos en ontologías locales);

c. Definir las fuentes externas que serán disponibles a los usuarios de la SRIA.
d. Definir las reglas de mapeo entre los elementos de la ontología y los datos definidos en el modelo de Dominio;
e. Definir operaciones entre las instancias de la ontología (p.e., para filtrar búsquedas de instancias externas).

3. **Modelo de navegación extendido** (*Extended Navigational Model*, ENM; modelo independiente de la plataforma). El modelo de navegación extendido es una extensión del modelo de navegación de OOH4RIA. El ENM especifica la forma por la cual los usuarios son capaces de acceder a los datos y a las instancias de ontología de la aplicación, definidas en los dos primeros modelos. Para cada rol de usuario de la aplicación, es posible definir un modelo de navegación diferente que filtre la información del servidor y los servicios que pueden ser invocados. El ENM también captura la forma en la que las SRIA publican su propio conocimiento de forma estructurada y conectan su información con otras fuentes de conocimiento de la Web.
4. **Modelo de presentación extendido** (*Extended Presentation Model*, EPM, modelo específico de la plataforma). El EPM extiende el modelo de presentación de OOH4RIA. Este modelo define la estructura del cliente de la SRIA representando las pantallas, paneles y widgets de la interfaz de usuario así como sus principales características: posición, tamaño y estilo (fuente, color, color de fondo, etc.). En contraste con el resto de modelos, este modelo es WYSIWYG, en el cual la visualización de la interaz de usuario es completamente equivalente a la de la interfaz posteriormente generada. En este modelo es posible incluir anotaciones basadas en las ontologías de dominio sobre los elementos estáticos de la interfaz de usuario.
5. **Modelo de orquestación extendido** (*Extended Orchestration Model*, EOM; modelo específico de la plataforma). El EOM es una extensión del modelo de orquestación de OOH4RIA. El EOM está representado como una colección de reglas Evento-Condición-Acción que especifican el comportamiento de la interfaz de usuario

acorde con los eventos producidos por los usuarios durante la interacción con la interfaz de usuario. Este modelo conecta los eventos producidos en la interfaz con las acciones que puede realizar el servidor, especificadas en el EDM y ENM.

6. **Modelo de visualización ontológico** (*Visualisation Ontology Model*, VOM; modelo independiente de la plataforma). El modelo de visualización ontológico combina el conocimiento contenido en el EPM y en el EOM para crear las instancias de la ontología de visualización, previstas en el modelo de anotación para las SRIA. Este modelo debería ser creado automáticamente por medio de una transformación modelo a modelo.

H.3.3. El proceso de desarrollo S^{m}4RIA

De forma similar a OOH4RIA, el proceso de desarrollo S^{m}4RIA esta dividido en tres actividades principales, que agrupan tareas y elementos de modelado con una misma finalidad:

1. Diseñar los elementos del servidor SRIA;
2. Diseñar los elementos del cliente SRIA; y
3. Generar la aplicación SRIA por medio de un conjunto de transformaciones modelo a texto.

El proceso de desarrollo S^{m}4RIA comienza con el diseño del servidor SRIA. En esta actividad, los diseñadores modelan todos los aspectos que serán usados durante el proceso de generación de la última actividad. La primera tarea de esta actividad, llevada a cabo por el diseñador del servidor, es la definición del modelo de Dominio, que, como se ha descrito en el apartado anterior, especifica las principales estructuras de datos de la aplicación, las relaciones entre ellas y las operaciones que se pueden aplicar sobre las mismas. A continuación, el diseñador de ontologías crea el EDM, que contiene la ontología de dominio, define que ontologías y bases de conocimiento externas son importadas y mapea las instancias de ontología y las estructuras de datos de la SRIA.

Ambos modelos de dominio son la entrada de la tarea de definición del modelo de navegación, en la que el diseñador del servidor especifica la forma en la cual los usuarios serán capaces de navegar por las estructuras de datos y las instancias de ontologías (locales o externas). Asimismo, el diseñador define que operaciones del servidor podrán ser invocadas por la interfaz de usuario o por clientes externos. En esta tarea el diseñador también concreta que servicios externos de la Web Semántica podrán ser invocados, previamente definidos en el EDM.

En la segunda actividad, se diseña el cliente SRIA y la interfaz de usuario. En una primera fase, el modelo de navegación extendido se transforma en el EPM de forma automática, y posteriormente, en el EOM por medio de dos transformaciones modelo a modelo llamadas *Nav2Pres* y *Nav&Pres2Orch*. Estas transformaciones son opcionales y crean el esqueleto de ambos modelos, que tendrá que ser completado por el diseñador de la interfaz. A continuación, en una segunda fase, el diseñador de ontologías (o el diseñador de la interfaz, dependiendo de su perfil) puede incluir anotaciones semánticas en la interfaz y enlazar los elementos de la interfaz con las fuentes de conocimiento externas, definidas en el EDM. Una vez completados el EPM y el EOM, el modelo ontológico de visualización es generado a partir de la información contenida en ellos por medio de la transformación modelo a modelo *Pres&Orch2Visu*, combinando de esta forma información acerca de la estructura y del comportamiento de la interfaz.

Finalmente, en la última actividad del proceso, los modulos software de la SRIA son generados por medio de un conjunto de transformaciones modelo a texto a partir de los modelos obtenidos en las dos primeras actividades. Estos procesos no pueden generar toda la aplicación a partir de los modelos, p.e., las operaciones personalizadas no pueden ser generadas automáticamente. Parte del código generado tendrá que ser completado por los desarrolladores.

H.4. S^{m}4RIA EXTENSION FOR OIDE

Para evaluar la metodología S^{m}4RIA y facilitar a su vez su adopción, todos sus elementos fueron implementados como una extensión de la herramienta software OIDE[54] llamada *S^{m}4RIA Extension for OIDE* (Hermida, Meliá, J.-J. Martinez, et al. 2012; Hermida, Meliá, Montoyo, et al. 2012b), que es la última contribución de esta tesis. Esta herramienta implementa los modelos S^{m}4RIA y automatiza los procesos de

[54] *OOH4RIA Integrated Development Environment*, Entorno de Desarrollo Integrado OOH4RIA

transformación (modelo a modelo y modelo a texto) necesarios para la generación de SRIAs.

OIDE es una aplicación basada en el entorno Eclipse, es decir, está desarrollada como un conjunto de plugins de Eclipse (la Figure H.19 muestra la pantalla principal de la aplicación). Esta aplicación define los metamodelos OOH4RIA usando el metamodelo EMOF y su sintaxis gráfica concreta por medio de los framework EMF y GMF. En OIDE los modelos de presentación y orquestación están integrados en un único modelo: el modelo de presentación OIDE.

La extensión definida sobre esta herramienta fue desarrollada para evaluar de forma cualitativa la validez de la metodología S^{m}4RIA. Para ello, en cada ciclo de desarrollo de la herramienta, se desarrolló uno de los casos de estudio para detectar posibles carencias o inconvenientes del método. Cuando el desarrollo se encontraba en sus etapas finales, la herramienta fue evaluada externamente en dos foros (uno nacional y el otro internacional), en los cuales las opiniones vertidas por los expertos fueron tomadas en consideración para refinar la metodología y la herramienta software que la implementa.

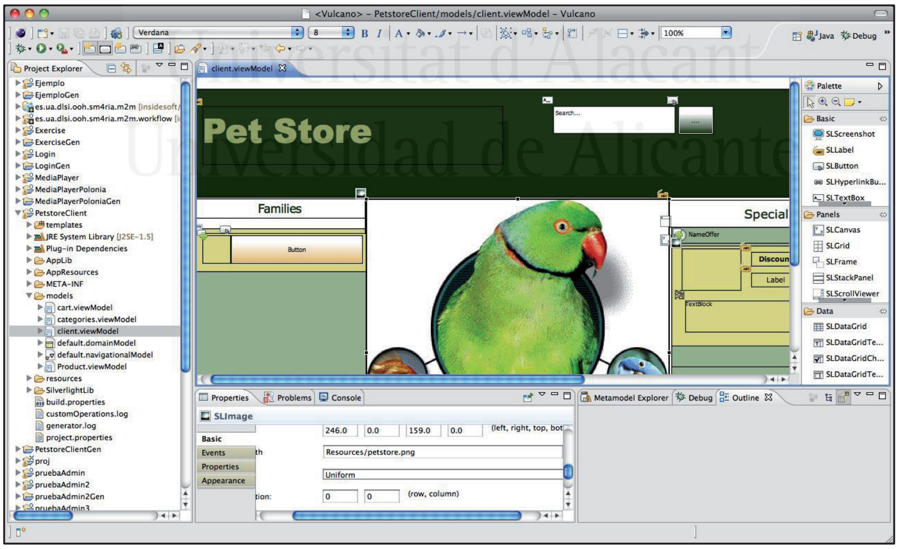


Figure H.19. Pantalla principal de la herramienta OIDE y de la extensión para S^{m}4RIA.

Esta herramienta implementa los editores de los modelos S^{m}4RIA, las reglas de transformación entre modelos y los flujos de trabajo que gestionan la generación de las aplicaciones SRIA utilizando los frameworks que provee el entorno Eclipse (p.e., EMF, GMF, Xtext, Xpand, QVT operational o MWE). Los siguientes apartados describen brevemente las nuevas funcionalidades y los nuevos elementos incorporados en la herramienta.

H.4.1. EDITORES DE MODELOS

Dos nuevos editores de modelos han sido implementados: el editor del modelo extendido de dominio y el editor del modelo ontológico de visualización; y dos editores han sido extendidos a partir de la implementación para la metodología OOH4RIA: el editor para el modelo de navegación extendido y el editor del modelo de presentación OIDE. Además, nuevos asistentes han sido desarrollados para ayudar a los usuarios en la creación de los modelos. Los editores incluidos en la herramienta son los siguientes:

- ***Extended Domain Model***, desarrollando utilizando el framework EMF (para la sintaxis abstracta y concreta del metamodelo) y Xtext (para implementar una segunda sintaxis concreta del metamodelo).
- ***Extended Navigational Model***, desarrollado utilizando los framework EMF (para la sintaxis abstracta del metamodelo) y GMF (para la sintaxis concreta del metamodelo.
- ***Extended OIDE Presentation Model***, desarrollado utilizando los framework EMF (para la sintaxis abstracta del metamodelo) y GMF (para la sintaxis concreta del metamodelo.
- ***Visualisation Ontology Model***, desarrollado utilizando los framework EMF (para la sintaxis abstracta y concreta del metamodelo).
- ***OOH4RIA Domain Model***. Reusado de OIDE sin modificación.

H.4.2. TRANSFORMACIONES

S^{m}4RIA extension for OIDE implementa todos los procesos de transformacion del proceso de desarrollo S^{m}4RIA:

- **Transformaciones modelo a texto**. Para generar los componentes software de las SRIA, esta extensión incluye los procesos de transformación que se definen en la tercera actividad del proceso

S^{m}4RIA. Para definir las reglas de transformación utiliza el lenguaje Xpand, que es interpretado por el motor de transformaciones Xpand de Eclipse.

- **Transformaciones modelo a modelo**. La herramienta implementa las reglas de transformación definidas en QVT operational y las ejecuta utilizando el motor de transformaciones QVTo de Eclipse. Las transformaciones incluidas en esta herramienta son las siguientes:
    - Transformación *Domain2EDM*: Modelo de dominio – modelo de dominio extendido.
    - Transformación *EDM2Domain* (beta): Modelo de dominio extendido – Modelo de dominio.
    - Transformación *EDM2ENM*: Modelo de dominio extendido – Modelo de navegación extendido.
    - Transformación *Navigation2Presentation*: Modelo de navegación extendido – Modelo de presentación.

H.4.3. NUEVOS PROCESOS

Los nuevos artefactos y procesos presentados en los dos apartados anteriores facilitan la adaptacion de la metodologia S^{m}4RIA a nuevos procesos de modernización y generación. Los procesos más relevantes desarrollados son los siguientes:

- *Generación automática de interfaces de usuario para administradores.* Usando las transformaciones modelo a modelo ya implementadas es posible generar automáticamente (o al menos la mayor parte de los módulos software) de una aplicación SRIA a partir del modelo de dominio o del modelo de dominio extendido.
- *Generación de interfaces RIA para fuentes de Linked Data.* Por medio de dos nuevas transformaciones texto a modelo y modelo a modelo que obtienen el modelo de dominio extendido a partir de una ontología OWL, es posible especificar un servidor RIA que gestione los datos externos de una fuente de Linked Data así como un cliente RIA que los visualice.

H.5. Conclusiones y trabajo futuro

H.5.1. Conclusiones

En base a los contenidos de esta tesis y a las contribuciones que se han presentado, este apartado trata de contestar las preguntas planteadas en la introducción, formuladas a partir de los problemas detectados.

RQ1 *– ¿Es posible mejorar la interoperabilidad de las Rich Internet Applications con otros sistemas software (como, por ejemplo, los buscadores Web) usando técnicas, tecnologías y recursos de la Web Semántica?*

Esta pregunta de investigación fue afirmativamente contestada con la propuesta de *Sematic Rich Internet Application*, presentada en el segundo apartado de este anexo. Las SRIAs han sido diseñadas como una extensión de las RIA tradicionales que utilizan las tecnologías de la Web Semántica para representar y compartir el conocimiento que utilizan en la Web. El uso de los principios de Linked Data, que son un estándar de-facto soportado por el W3C, y las tecnologías para representar y compartir conocimiento basadas en la representación de ontologías (en OWL) facilitan que el conocimiento compartido pueda ser fácilmente reutilizado. Esta aproximación permite que los clientes Web puedan acceder a todo el contenido de la SRIA de una forma independiente a la tecnología de implementación de la SRIA, solventando la principal limitación de las RIA tradicionales.

La principal limitación de la solución es que para su diseño no se han considerado aspectos relacionados con el rendimiento de la aplicación en escenarios reales con diferentes cargas de trabajo. A pesar de que para la evaluación de la plataforma SRIA se utilizaron casos de estudio reales, sería necesario medir empíricamente el rendimiento de la aplicación bajo diferentes condiciones y consultas de información para validar la arquitectura software propuesta.

Otro aspecto no evaluado de forma empírica en esta tesis es el beneficio del modelo de anotación propuesto para la generación de anotaciones semánticas en las SRIA.

RQ2 *– ¿Cómo pueden las actuales metodologías dirigidas por modelos ser extendidas para desarrollar la solución propuesta a los problemas detectados en las Rich Internet Applications?*

Para contestar a esta segunda pregunta de investigación, se diseñó la metodología S^{m}4RIA, como una extensión de la metodología OOH4RIA, para el desarrollo de la propuesta SRIA. La aproximación S^{m}4RIA introduce un conjunto de artefactos y procesos para el desarrollo de SRIAs, incorporando las primitivas necesarias para modelar los componentes de la SRIA que permiten representar y compartir conocimiento en la Web.

El desarrollo de los casos de estudio propuestos y el uso de la metodología en dos proyectos, que siguen en marcha, ha demostrado los potenciales beneficios de la metodología:

Dado que es una metodología de desarrollo dirigido por modelos: (a) reduce el coste de desarrollo y mantenimiento de las SRIA y (b) facilita la definición de la aplicación SRIA completa en tiempo de diseño;

(c) facilita que desarrolladores no expertos utilicen las bases de conocimiento disponibles en la Web y puedan crear nuevas;

(d) simplifica la creación y explotación de servicios de Linked Data en RIA. La simplicidad de los procesos y los modelos definidos deberían reducir la curva de aprendizaje y permitir que diseñadores no expertos puedan utilizar las tecnologías de forma sencilla.

RQ3 *– ¿Cómo se pueden implementar las soluciones propuestas en una herramienta software CASE?*

Esta última pregunta de investigación fue contestada con la implementación de la metodología S^{m}4RIA en la herramienta software llamada *S^{m}4RIA extension for OIDE*. Esta herramienta extiende las funcionalidades de la herramienta OIDE, que implementa la metodología OOH4RIA. El hecho de reusar una herramienta existente y el framework para modelado de Eclipse facilitó el desarrollo de los modelos y las reglas de transformación.

La herramienta fue evaluada de forma iterativa conjuntamente con los elementos de la metodología con el desarrollo de los casos de estudio.

Para resumir las contribuciones de esta tesis, la Table H.10 asocia los objetivos introducidos en el primer apartado de este resumen, planteados a partir de las preguntas de investigación, y las tareas realizadas para poder cumplirlos.

Table H.10. Resumen de las tareas y contribuciones realizadas asociadas a cada objetivo.

Pregunta de investigación	Objetivo	Descripción	Tareas realizadas
RQ1	O1	Mejorar la interoperabilidad de las Rich Internet Applications con los sistemas de la Web que usan el texto como entrada (e.g., buscadores o lectores para invidentes).	Desarrollo de una propuesta de *Semantic Rich Internet Application*.
	O1.1	Mejorar la exportabilidad de los datos contenidos en Rich Internet Applications.	• Las SRIA incluyen nuevos módulos software en su servidor para compartir información como Linked Data (p.e., la base de conocimiento o el servicio SPARQL) de una forma estándar. • Las SRIA también pueden incluir interfaces HTML con anotaciones semánticas basadas en el modelo de anotación propuesto.
	O1.2	Mejorar el acceso a la información relativa a los elementos multimedia.	Las ontologías de dominio y visualización, incluidas en el modelo de anotacion propuesto para las SRIA, pueden ser usadas para compartir información de los elementos multimedia por medio de los módulos software.
	O1.3	Reusar técnicas, tecnologías y recursos ya existentes en la Web Semántica.	• La propuesta SRIA reutiliza parte de la arquitectura de la Web Semántica y los principios de Linked Data para compartir datos por medio de Internet. • El uso de ontologías para representar el conocimiento utilizado por una SRIA. • El uso de los lenguajes estándar OWL, RDF y

Pregunta de investigación	Objetivo	Descripción	Tareas realizadas
			SPARQL para la representación de ontologías, instancias de ontologías y consultas a las bases de conocimiento, respectivamente.
	O1.4	Desarrollar una colección de casos de estudio para evaluar la validez de la solución propuesta.	• Desarrollo de un reproductor multimedia como una SRIA. • Desarrollo de una red social como una SRIA. • Desarrollo de una red social para empresas como una SRIA.
RQ2	O2	Diseñar una metodología de desarrollo software dirigido por modelos para el desarrollo de la solución propuesta.	Desarrollo de la metodología S^{m}4RIA dirigida por modelos para el desarrollo de SRIA.
	O2.1	Facilitar el desarrollo de la solución propuesta en O1.	Diseño de nuevos modelos en S^{m}4RIA adaptados a las nuevas características de las SRIA: • El modelo extendido de dominio, para el diseño de ontologías de dominio. • El modelo extendido de navegación, para especificar la forma en que las instancias de la ontología son utilizadas por la aplicación. • Los modelos de presentación y orquestación extendidos, para la visualización de las instancias importadas desde fuentes externas. Diseño de una colección de transformaciones modelo a modelo que acelera la creación de los esqueletos de los modelos para los diseñadores.
	O2.2	Mejorar la mantenibilidad de la solución propuesta en O1.	Al ser una metodología dirigida por modelos, los cambios en los requerimientos de una aplicación sólo implicarían modificaciones en los modelos y la regeneración del código fuente. El framework Xpand protege el código personalizado cuando se reinvocan las transformaciones modelo a texto.

Pregunta de investigación	Objetivo	Descripción	Tareas realizadas
	O2.3	Extender una metodología existente para el desarrollo de RIA.	Diseñar S^{m}4RIA como una extensión de OOH4RIA, especializada en el desarrollo de RIA.
RQ3	O3	Implementar los elementos de la metodología diseñada en una herramienta software CASE.	Desarrollo de una herramienta software llamada *S^{m}4RIA extension for OIDE*, que implementa S^{m}4RIA. Evaluar la herramienta en reuniones científicas.

H.5.2. TRABAJO FUTURO

A partir de las conclusiones obtenidas, en este último apartado, se definen las principales líneas de trabajo futuro. Relacionadas con el campo de la Web Semántica, las siguientes líneas de trabajo permanecen abiertas:

- Evaluación empírica del modelo de anotación propuesto utilizando un motor de búsqueda de la Web Semántica.
- Extender un cliente software de la Web Semántica para explotar el modelo de anotación propuesto. Los motores de búsqueda actuales no se ajustan totalmente a las características del modelo propuesto.
- Evaluación empírica de la arquitectura de la SRIA basada en parámetros relacionados con el rendimiento.
- Repetir los procesos de evaluación con la propuesta SRIA para el campo de la Inteligencia de Negocio.

Relacionadas con el campo de la Ingeniería Web, las líneas de investigación futura son las siguientes:

- Evaluación empírica de los elementos de la metodología S^{m}4RIA. El objetivo es analizar diferentes parámetros relacionados con la usabilidad y la mantenibilidad de los modelos con un grupo real de desarrolladores con el objetivo de detectar limitaciones de la metodología y facilitar su adopción en escenarios de empresa. Estos experimentos ya han comenzado

- Adaptar el modelo de arquitectura de OOH4RIA a S^{m}4RIA para poder personalizar la arquitectura de las SRIA generadas.
- Continuar el estudio de los procesos de modernización para la generación automática de interfaces RIA. Este estudio se centrará en los métodos de visualización de datos y en la adaptación de las transformaciones modelo a modelo.
- Estudiar los procesos de modernización para modelar aplicaciones para móviles o intefaces Web que hagan uso de la información de las SRIA.
- Estudiar la definición de líneas de producto para la generación de SRIA. Estas líneas podrían personalizar la generación a partir de unas opciones predefinidas.
- Estudiar las ventajas y las limitaciones de los modelos con una sintaxis concreta gráfica respecto a aquellos con una sintaxis concreta textual.

Finalmente, existen algunas cuestiones relacionadas con la herramienta software desarrollada que deberían ser estudiadas o completadas:

- Completar la implementación de las transformaciones modelo a texto para el caso de estudio de SRIA para Inteligencia de Negocio.
- Completar la implementación de los procesos de modernización explicados.
- Mejorar la usabilidad general de la herramienta. A parte de comprobar que las funcionalidades necesarias han sido implementadas, los aspectos relacionados con la usabilidad de la herramienta deberían ser estudiados e implementar las mejoras necesarias en el proceso de diseño y generación.
- Mejorar la usabilidad de los editores de los modelos para mejorar la eficiencia de los desarrolladores y la integración entre los editores de modelos y los procesos de transformación.