UNIVERSITAT POLITÈCNICA DE VALÈNCIA

OOWS 2.0

Un Método de Ingeniería Web Para la Producción de Aplicaciones Web 2.0

Tesis Doctoral de
Francisco Valverde Giromé

Director
Oscar Pástor López

PROS
Centro de Investigación en Métodos de Producción de Software

Octubre del 2010

UNIVERSIDAD POLITECNICA DE VALENCIA

PROS

Centro de Investigación en Métodos de Producción de Software

Tesis Doctoral

OOWS 2.0: Un Método de Ingeniería Web Dirigido por Modelos para la Producción de Aplicaciones Web 2.0

Francisco Valverde Giromé

Director: Óscar Pastor López

Octubre del 2010, Valencia

OOWS 2.0: Un Método de Ingeniería Web Dirigido por Modelos para la Producción de Aplicaciones Web 2.0

OOWS 2.0: A Model-driven Web Engineering Method for the Production of Web 2.0 Applications

OOWS 2.0: Un Mètode d'Enginyeria Web Dirigit per Models per a la Producció d'Aplicacions Web 2.0

Tesis defendida por Francisco Valverde Giromé el 26 de octubre del 2010 para la obtención del título de Doctor en Informática por la Universidad Politécnica de Valencia

Director

Dr. Oscar Pástor López, *Universidad Politécnica de Valencia*

Tribunal de tesis

Dr. Vicente Pelechano, *Universidad Politécnica de Valencia*

Dra. Paloma Martínez, *Universidad Politécnica de Madrid*

Dr. Gustavo Hector Rossi, *Universidad Nacional de La Plata*

Dr. Philippe Palanque, *Université Paul Sabatier - Toulouse III*

Dr. Sven Casteleyn, *Vrije Universiteit Brussel*

A Ordino y La Laguna Grande,
"Porque solamente estando en lo más profundo del valle,
puede saberse lo magnífico que es estar
en la cima de una montaña"

Agradecimientos

Decía mi abuela que "Es de mal nacido, el ser desagradecido" y en la confección de esta tesis, hay mucha gente a la que le debo, como mínimo, un agradecimiento. Además, esta es la única sección que casi todo el mundo lee para ver si está incluido. Así que doy las gracias:

… a quien me dio la oportunidad de entrar en el grupo OO-Method y mostrarme el camino de la investigación

… a quien me enseñó que siempre hay que mirar hacia delante y si te pierdes es mejor preguntar que mirar el GPS

… a quienes padecieron conmigo tanto la escritura de *papers* como mis correcciones

… a los miembros del Comité Porrero del ProS y a aquéllos que financiaron su actividad con sus perdidas

… a quienes con su compañía hicieron agradables los "deliciosos" manjares de La Vella

… al defensor quijotesco de la ambigüedad del concepto de CIM

… a quienes corrieron conmigo a lo largo de múltiples carreras populares

… al creyente del roelismo y seguidor del pequeño saltamontes

… al invencible ProS F.C

… a mi voz de la conciencia y asesor de burocracia doctoral

… a quien me alimentó durante estos años gracias a sus *sándwiches* y *tupperwares*

… al Ministerio de Educación y Ciencia, por otorgarme mi sustento económico en forma de una beca FPU (AP2005-1590)

… a quien me sugirió que lo que proponía lo llamaban *weaving models*

… al matrimonio de la eterna sonrisa

… a quienes sufrieron, durante reuniones maratonianas, el frío polar del seminario 3S02

… a la defensora de la palabra deleción

… a quienes no les importó jugar al pádel a las 8 de la mañana

… a quienes tuvieron fe que un equipo sudamericano llegaría a semifinales del mundial, aunque me costase 30€

… a quienes compartieron conmigo los laboratorios L04 y se marcharon en busca de una vida mejor

… a quienes al llegar a casa se preocuparon por mi

Y sobre todo, a quien me ha ayudado tanto durante la escritura de esta tesis que no me basta una línea para darle las gracias.

Francisco Valverde

Prefacio

Esta tesis doctoral es el final de un largo camino que empecé hace cinco años, buscando un proyecto de final de carrera con una única cosa en la cabeza: hacer algo relacionado con la Web. Lo que no sabía es que esa búsqueda acabaría convirtiéndose en mi trabajo como investigador. Mi investigación con el método OOWS establece una trilogía: la primera parte fue su aplicación en mi PFC, la segunda parte su implementación en mi tesis de Máster y, por último, su evolución como OOWS 2.0, en la presente tesis doctoral. El hecho de trabajar con este método me ha permitido, no solo conocer a fondo los detalles del desarrollo Web, algo que siempre me ha gustado, sino además poner en práctica mis propias ideas. La esencia de estas ideas es mi pequeña aportación al vasto e infinito universo de conocimiento de la red de redes.

Resumen

Los métodos de Ingeniería Web dirigidos por modelos han mejorado tanto la calidad como la eficiencia, a la hora de desarrollar aplicaciones Web. Estos métodos utilizan modelos conceptuales para capturar, de manera abstracta, una representación detallada de la aplicación Web a desarrollar. La ventaja más destacada de esta aproximación es que a partir de estos modelos, ampliamente validados en entornos industriales, es factible la generación sistemática del código que implementa la aplicación Web.

Las aplicaciones Web 2.0 destacan, fundamentalmente, por la alta implicación de los usuarios a la hora de crear sus contenidos, ya sean en forma de opiniones, fotos, definiciones o videos. Este hecho enfatiza el carácter colaborativo de las aplicaciones Web 2.0 y, otorga al usuario un rol esencial en la aplicación. Para lograr la colaboración del usuario, es un requisito imprescindible que la aplicación proporcione una interacción precisa e intuitiva. Este objetivo se ha conseguido en las aplicaciones Web 2.0 mediante: (1) una interfaz de usuario tecnológicamente avanzada y con un elevado grado de usabilidad; (2) la reutilización de un conjunto de buenas prácticas, ampliamente aplicadas en las aplicaciones Web 2.0, que proporcionan al usuario un modo sencillo y conocido de interactuar con la aplicación. Con el objetivo de desarrollar aplicaciones Web 2.0 de calidad, ambos aspectos tienen que estar presentes en los métodos de Ingeniería Web.

Esta tesis doctoral presenta el método de Ingeniería Web dirigido por modelos OOWS 2.0 como una evolución incremental y necesaria del método OOWS. La contribución principal de este nuevo método es proporcionar la expresividad conceptual requerida para el desarrollo de aplicaciones Web 2.0. En concreto, las contribuciones se centran en soportar los aspectos avanzados de la interacción con el usuario final, tan relevantes en dichas aplicaciones. Para lograr esta meta, la tesis doctoral introduce una serie de modelos conceptuales que capturan, sin ambigüedades, las nuevas necesidades de interacción demandadas por las aplicaciones Web 2.0.

En primer lugar, la tesis doctoral presenta un Modelo de Interacción Abstracto para describir la interacción independientemente de la tecnología utilizada. Este modelo emerge a partir de las experiencias satisfactorias obtenidas con los métodos OO-Method y OOWS. Mediante la selección y extensión de las primitivas conceptuales de ambos métodos, se proporciona un nuevo modelo con la expresividad necesaria para definir la interacción de las aplicaciones Web 2.0.

Con el objetivo de modelar interfaces que enriquezcan la experiencia del usuario final, la tesis introduce un Modelo de Interfaces de Usuario para *Rich Internet Applications* (RIA). Las tecnologías RIA son un elemento clave en las aplicaciones Web 2.0 para definir interfaces con un elevado nivel de usabilidad. El modelo propuesto soporta el modelado de interfaces de usuario avanzadas integrando en el método, el amplio abanico de tecnologías RIA disponibles.

Por último, la tesis doctoral introduce el concepto de patrón Web 2.0 en el nivel de modelado conceptual. Un patrón Web 2.0 representa un mecanismo, recurrentemente utilizado en el desarrollo de aplicaciones Web 2.0, con el objetivo de mejorar la interacción con el usuario. La tesis define, a partir de un conjunto relevante de aplicaciones Web 2.0, un catálogo de patrones Web 2.0 y cómo especificarlos usando modelos conceptuales. Asimismo, se propone una estrategia basada en transformaciones entre modelos, con el fin de introducir la semántica de estos patrones en un método de Ingeniería Web.

Estas propuestas son integradas y aplicadas rigurosamente en el marco del método OOWS 2.0. Como resultado final se presenta un método original de Ingeniería Web dirigido por modelos que soporta, de forma integral, el modelado avanzado de la interacción para aplicaciones Web 2.0.

Resum

Els mètodes d'Enginyeria Web dirigits per models han millorat tant la qualitat com la eficiència a l'hora de desenvolupar aplicacions Web. Aquests mètodes utilitzen models conceptuals per a capturar, de manera abstracta, una representació detallada de l'aplicació Web a desenvolupar. L'avantatge més destacat d'aquesta aproximació és que a partir d'aquests models, àmpliament validats als entorns industrials, és factible la generació sistemàtica del codi que implementa l'aplicació Web.

Les aplicacions Web 2.0 destaquen, fonamentalment, per l'alta implicació dels usuaris a l'hora de crear el seus continguts, ja siguen en forma d'opinions, fotos, definicions o vídeos. Aquest fet emfatitza el caràcter col·laboratiu de les aplicacions Web 2.0 i, atorga a l'usuari un rol primordial en l'aplicació. Per tal d'aconseguir la col·laboració de l'usuari, és un requisit imprescindible que l'aplicació proporcione una interacció precisa i intuïtiva. Aquest objectiu s'ha aconseguit a les aplicacions Web 2.0 mitjançant: 1) una interfície d'usuari avançada tecnològicament i amb un elevat grau d'usabilitat; 2) la reutilització d'un conjunt de bones pràctiques, àmpliament aplicades al domini de la Web 2.0, que proporcionen a l'usuari una forma senzilla i coneguda d'interactuar amb l'aplicació. Amb l'objectiu de desenvolupar aplicacions Web 2.0 de qualitat, ambdós aspectes han d'estar presents als mètodes d'Enginyeria Web.

Aquesta tesis doctoral presenta el mètode d'Enginyeria Web dirigit per models OOWS 2.0 com una evolució incremental i necessària del mètode OOWS. La contribució principal d'aquest nou mètode és proporcionar l'expressivitat conceptual requerida per al desenvolupament d'aplicacions Web 2.0. En concret, les contribucions es centren en suportar els aspectes avançats de l'interacció amb l'usuari final, tan rellevants en aquest tipus d'aplicacions. Per tal d'aconseguir aquesta meta, la tesis doctoral introdueix una sèrie de models conceptuals que representen, sense ambigüitats, les noves necessitats d'interacció demandades per les aplicacions Web 2.0.

En primer lloc, la tesis doctoral presenta un Model d'Interacció Abstracte per a descriure l'interacció, independentment, de la tecnologia utilitzada. Aquest model emergeix a partir de les experiències satisfactòries obtingudes amb els mètodes OO-Method i OOWS. Mitjançant la selecció i extensió de les primitives conceptuals d'ambdós mètodes, es proporciona un nou model amb l'expressivitat necessària per a definir l'interacció de les aplicacions Web 2.0.

Amb el propòsit de modelar interfícies que enriquesquen l'experiència de l'usuari final, la tesis introdueix un Model d'Interfícies d'Usuari per a *Rich Internet Aplications* (RIA). Les tecnologíes RIA són un element clau, a les aplicacions Web 2.0, per definir interfícies amb un elevat grau d'usabilitat. El model proposat suporta el modelat d'aquestes interfícies d'usuari avançades, integrant al mètode l'ampla sèrie de tecnologíes RIA disponibles.

Per últim, la tesis introdueix el concepte de patró Web 2.0 a nivell de modelat conceptual. Un patró Web 2.0 representa un mecanisme, recurrentment utilitzat a l'hora de desenvolupar aplicacions Web 2.0, amb la finalitat de millorar l'interacció amb l'usuari. La tesis defineix, a partir d'un conjunt rellevant d'aplicacions Web 2.0 reals, un catàleg de patrons Web 2.0 i com representar-los utilitzant models conceptuals. Tanmateix, es proposa una estratègia basada en transformacions entre models, amb la fi d'introduir la semàntica d'aquestos patrons en un mètode d'Enginyeria Web.

Aquestes propostes són integrades i aplicades rigorosament al marc del mètode OOWS 2.0. Com a resultat final es presenta un mètode original d'Enginyeria Web dirigit per models que suporta, de forma integral, el modelat avançat de l'interacció a les aplicacions Web 2.0.

Overview

Model-driven Web Engineering methods have improved both the quality and the efficiency of the Web application development. These methods use conceptual models for gathering, in an abstract way, a detailed representation of the Web application to be developed. The most notable advantage of this approach is that using these models, which have been widely validated in industrial environments, it is feasible the systematic code generation of a Web application.

Web 2.0 applications emphasize a high user involvement in order to create the application content: personal opinions, photos, definitions or video. This fact highlights the collaborative facet of Web 2.0 applications and the fundamental role of the end-user. With the goal of achieving the end-user collaboration, it is an essential requirement the definition of a precise and intuitive interaction process. This goal has been achieved in Web 2.0 applications by means of: 1) an advanced user interface, from the technological point of view, and with a high usability degree; 2) the reuse of the best practices, which have been widely applied into the Web 2.0 applications. These practices provide to the user a well-known and precise way to interact with the application. In order to develop Web 2.0 quality applications, both characteristics must be included into Web Engineering methods.

This PhD thesis presents the OOWS 2.0 model-driven Web Engineering method as an incremental and required evolution of the OOWS method. The main contribution of this new method is to provide the required conceptual expressivity for developing Web 2.0 applications. Specifically, the contributions are focus on supporting the advanced interaction aspects, which are very important in this domain, with end-users. With the purpose of achieving this goal, the PhD thesis introduces a set of conceptual models, which gather, unambiguously, the new interaction requirements demanded by Web 2.0 applications.

First, the PhD thesis presents an Abstract Interaction Model for describing the interaction from a technologically independent point of view. This

model arises from the lessons learned in the application of the OOWS and OO-Method methods. This new model is provided by means of the selection and extension of conceptual primitives from both methods. The result is a model to specify precisely the interaction process of a Web 2.0 application.

With the goal of enriching the user experience using the interface, the PhD thesis introduces a User Interface Model for Rich Internet Applications (RIA). RIA technologies are a key element in Web 2.0 applications in order to develop user interfaces with a high usability level. The proposed model supports the wide range of RIA technologies available to deal with the user interface modelling.

Finally, the PhD thesis introduces the Web 2.0 pattern term at the modelling level. A Web 2.0 pattern represents a mechanism, recurrently applied in the Web 2.0 development, with the goal of improving the end-user interaction. The PhD thesis defines, from a set of popular Web 2.0 applications, a Web 2.0 pattern library. For providing a formal description, the patterns of this library are represented using a strategy based on conceptual models. Furthermore, a model-to-model transformation strategy is proposed in order to introduce the patterns semantics into a Web Engineering method.

These proposals are rigorously integrated and applied in the context of the OOWS 2.0 method. As final result, the most significant contribution of this PhD thesis is an original model-driven Web Engineering method, which supports the advanced interaction required by Web 2.0 applications.

Índice

1 Introducción 1

1.1 ¿Qué son las aplicaciones Web 2.0? 1

1.2 Motivación 5

1.3 Objetivos de la tesis 8

1.4 Solución propuesta 9

1.5 Ámbito de aplicación 10

1.6 Estructura de la tesis doctoral 11

2 Fundamentos Metodológicos 15

2.1 Metodología de investigación: Design Science 15

2.2 Visión general del método OOWS y del método OOWS 2.0 20

2.3 El Modelo de Presentación de OO-Method 27

2.4 Los Modelos de Navegación y de Presentación de OOWS 32

2.5 Conclusiones 37

3 Estado del Arte 39

3.1 Trabajos relacionados con el modelado de la interacción 40

3.2 Trabajos relacionados en el ámbito de los métodos de Ingeniería Web .. 48

3.2.1 WEBML 48

3.2.2 RUX-Method *53*

3.2.3 OOH4RIA *56*

3.2.4 OOHDM *59*

3.2.5 UWE *62*

3.2.6 Análisis comparativo *63*

3.3 Trabajos relacionados en el desarrollo de RIA 68

3.4 Trabajos relacionados con la definición de Patrones para la Web 2.0 70

3.5 Conclusiones 74

4 Especificación del Modelo de Interacción Abstracto 75

4.1 Modelado de la interacción 75

4.2 Soporte a la interacción en OO-Method y OOWS 80

4.2.1 Recuperación de la información *80*

4.2.2 Ejecución de servicios *84*

4.2.3 Acceso a la aplicación y navegación *85*

4.3 Definición del Modelo de Interacción Abstracto 91

4.3.1 El Diagrama de Usuarios y el Mapa de Interacción *91*

4.3.2 Unidades de Interacción Abstractas *96*

4.3.3 Patrones Auxiliares de Interacción *100*

4.3.3.1 Filtro (Filter) 104

4.3.3.2 Índice (Index) 106

4.3.3.3 Vista Resumida (Summary View) 107

4.3.3.4 Ordenación (Order Criteria) 108

4.3.3.5 Paginación (Pagination) 110

4.3.3.6 Selección Enumerada (Defined List) 111

4.3.3.7 Regla de Validación (Validation Rule) 112

4.3.3.8 Inicialización de Argumento (Argument Setting) 114

4.3.3.9 Navegación por Objeto (Object Navigation) 116

4.3.3.10 Navegación Relacional (Relationship Navigation) 118

4.3.3.11 Navegación de Servicio (Service Navigation) 119

4.3.3.12 Filtro Navegacional (Navigation Filter) 121

4.3.3.13 Edición de Instancia (Instance Edition) 122

4.4 Conclusiones 125

5 Modelado de IU mediante Tecnologías RIA 129

5.1 ¿Qué son las Rich Internet Applications? 130

5.2 Definición de un metamodelo para el desarrollo de interfaces RIA 137

5.2.1 Modelado de los componentes de la Interfaz RIA 138

5.2.2 Modelado de la interacción dirigida por eventos de IU 149

5.3 Aplicación del metamodelo RIA en un método de Ingeniería Web 158

5.3.1 Estrategia de conexión a nivel de metamodelado 159

5.3.2 Modelado conceptual de la Interfaz RIA 168

5.4 Generación del código de la Interfaz RIA: *Adobe Flex* 171

5.5 Conclusiones 179

6 Patrones de Modelado para Aplicaciones Web 2.0 181

6.1 Análisis de patrones Web 2.0 182

6.2 Patrones Web 2.0: modelado conceptual 196

6.2.1 Quick Comment 202

6.2.2 Notification 205

6.2.3 Suggestion 208

6.2.4 Public Profile 211

6.2.5 Subscription 214

6.3 Incorporación en un método de Ingeniería Web 217

6.3.1 Estrategia propuesta 218

6.3.2 Aplicación en el método OOWS 2.0 223

6.3.2.1 Introducción del patrón Quick Comment 224

6.3.2.2 Introducción del patrón Notification 228

6.3.2.3 Especificación de las transformaciones de los patrones 232

6.4 Conclusiones 238

7 Evaluación de la Viabilidad 241

7.1 Introducción 242

7.2 Análisis del soporte proporcionado a nivel de modelado 243

7.3 Modelado de la demostración de laboratorio 251

7.3.1 Modelado del escenario Compare Genes 252

7.3.2 Modelado del escenario 23andMe Community 261

7.4 Diseño experimental para la evaluación del método 268

7.4.1 Definición 269

7.4.2 Planificación 271

7.4.3 Instrumentación 277

7.4.4 Proceso 282

7.5 Conclusiones 284

8 Conclusiones 287

8.1 Contribuciones principales 287

8.2 Trabajo actual y futuro 291

8.3 Publicaciones 294

8.4 Reflexión final 297

Bibliografía 299

Anexos 313

A.1 Ejemplo ilustrativo: *My Social Research* 313

A.2 Metamodelo de Interacción Abstracto 317

A.3 Metamodelo RIA para Adobe Flex 327

Índice de figuras

Fig. 2.1: *Framework* de investigación utilizado 17

Fig. 2.2: Metamodelo principal del método OOWS 22

Fig. 2.3: Fase de modelado conceptual en el método OOWS 24

Fig. 2.4: Fase de modelado conceptual del método OOWS 2.0 26

Fig. 2.5: Modelos Conceptuales de OOWS y OOWS 2.0 27

Fig. 2.6: Estructura del Modelo de Presentación OO-Method 28

Fig. 2.7: Editor del Modelo de Presentación en *OLIVANOVA* 32

Fig. 2.8: Ejemplo de Mapa de Navegación OOWS 33

Fig. 2.9: Ejemplo de la definición de una AIU en OOWS 34

Fig. 3.1: Ejemplo de CTT para la ejecución de un servicio 41

Fig. 3.2: *Framework* de referencia propuesto en el proyecto Chamaleon 46

Fig. 3.3: Patrón *Rating* en WebML según [Fraternali, Tisi *et al.* 2009] 52

Fig. 3.4: Proceso de desarrollo de RUX-Method 55

Fig. 3.5: Metamodelo de presentación de OOH4RIA 57

Fig. 3.6: Ejemplo del Modelo de Orquestación de OOH4RIA 58

Fig. 3.7: ADV definida en [Rossi, Urbieta *et al.* 2008] 61

Fig. 4.1: *User Class* para el agente "*Researcher*" 94

Fig. 4.2: Mapa de Interacción para el usuario *Researcher* 95

Fig. 4.3: Notación para la especificación de la *Population AIU* 97

Fig. 4.4: Ejemplo de una *Population AIU* 98

Fig. 4.5: Estructura del Modelo de Interacción Abstracto 103

Fig. 4.6: Ejemplo de un *Index AIP* 107

Fig. 4.7: Notación para el *Object Navigation AIP* 117

Fig. 4.8: Ejemplo de un *Object Navigation AIP* 118

Fig. 4.9: Ejemplo de un *Relationship Navigation AIP* 119

Fig. 4.10: Ejemplo de un *Navigation Filter AIP* 122

Fig. 4.11: Notación para el *Instance Edition AIP* 124

Fig. 4.12: Ejemplo de un *Instance Edition AIP* 125

Fig. 5.1: Ejemplos de distintos *widgets* en EXT-JS 141

Fig. 5.2: Vista de Interfaz del Metamodelo RIA 144

Fig. 5.3: Vista de *Event Rules* del Metamodelo RIA 152

Fig. 5.4: Gramática para la definición de *Event Rules* 157

Fig. 5.5: Ejemplo de la definición de una *Event Rule* 158

Fig. 5.6: *Core Weaving Metamodel* 161

Fig. 5.7: *Weaving Metamodel* para OOWS 2.0 162

Fig. 5.8: Relación entre los distintos niveles conceptuales 166

Fig. 5.9: Edición de un *weaving model* con Eclipse EMF 166

Fig. 5.10: Modelado conceptual de la Interfaz RIA en OOWS 2.0 169

Fig. 5.11: Ejemplo de código de IU en Flex 174

Fig. 5.12: Ejemplo de código generado en *ActionScript*. 175

Fig. 6.1: Ejemplo del patrón *Quick comment* 183

Fig. 6.2: Ejemplo del patrón *Tag definition* 184

Fig. 6.3: Ejemplo del patrón *Notification* 185

Fig. 6.4: Ejemplo del patrón *Collaborative Editing* 186

Fig. 6.5: Ejemplos del patrón *Quick Rating* 186

Fig. 6.6: Ejemplo del patrón *Reputation* 187

Fig. 6.7: Ejemplo del patrón *Share Content* 188

Fig. 6.8: Ejemplos del patrón *Suggestion* 188

Fig. 6.9: Ejemplo del patrón *Invite* 189

Fig. 6.10: Ejemplo del patrón *Public Profile* 190

Fig. 6.11: Ejemplo del patrón *Availability* 190

Fig. 6.12: Ejemplo del patrón *Ranking* 191

Fig. 6.13: Ejemplo del patrón *Favorite* 192

Fig. 6.14: Ejemplo del patrón *Subscription* 193

Fig. 6.15: Porcentaje de aplicación de cada patrón 195

Fig. 6.16: Modelos del patrón *Quick Comment* 204

Fig. 6.17: Modelos del patrón *Notification* 208

Fig. 6.18: Modelos para el patrón *Suggestion* 211

Fig. 6.19: Modelos para el patrón *Public Profile* 214

Fig. 6.20: Modelos para el patrón *Subscription* 217

Fig. 6.21: Proceso de generación del modelo del patrón Web 2.0 222

Fig. 6.22: Punto de extensión metodológico para el patrón *Quick Comment* 225

Fig. 6.23: Aplicación del patrón *Quick Comment* en OOWS 2.0 226

Fig. 6.24: Modelo resultante del patrón *Quick Comment* 228

Fig. 6.25: Punto de extensión metodológico para el patrón *Notification* 229

Fig. 6.26: Aplicación del patrón *Notification* en OOWS 230

Fig. 6.27: Modelo resultante del patrón *Notification* 232

Fig. 7.1: Visión General de *23andMe* 245

Fig. 7.2: Porcentajes de soporte a los escenarios 250

Fig. 7.3: Modelo de Objetos del escenario *Compare Genes* 253

Fig. 7.4: *Population AIU* del contexto *Compare Genes* 254

Fig. 7.5: *Service AIU* del contexto *Compare Genes* 255

Fig. 7.6: Interfaz RIA del escenario *Compare Genes* 258

Fig. 7.7: *Event Rule* para recalcular la comparación de características 259

Fig. 7.8: *Event Rules* para añadir una nueva característica a la comparación..... 260

Fig. 7.9: *Event Rule* para mostrar la información detallada 261

Fig. 7.10: Interfaz del escenario *23andMe Community* 262

Fig. 7.11: Modelo de Objetos del escenario *23andMe Community* 264

Fig. 7.12: Modelo de Interacción para el escenario *23andMe Community* 266

Fig. A.1: Vista Principal del Metamodelo RIA 318

Fig. A.2: Vista de la *Population AIU* 319

Fig. A.3: Vista de la *Service AIU* 320

Fig. A.4: Vista del *Argument Setting AIP* 321

Fig. A.5: Vista de los AIP *Pagination* y *Order Criteria* 322

Fig. A.6: Vista del *Filter AIP* 322

Fig. A.7: Vista de los AIP *Summary View* y *Index* 323

Fig. A.8: Vista de los AIP *Defined List* y *Validation Rule* 324

Fig. A.9: Vista del *Instance Edition AIP* 325

Fig. A.10: Vista de los AIP de Navegación 325

Fig. A.11: Vista del *Navigation Filter AIP* 326

Fig. A.12: Vista *Core* del Metamodelo RIA 327

Fig. A.13: Entidades *Layout* 328

Fig. A.14: Entidades *DataView* 329

Fig. A.15: Entidades *Input* (1) 329

Fig. A.16: Entidades *Input* (2) 330

Fig. A.17: Entidades *Service* y *Navigation* 330

Fig. A.18: Eventos de *Adobe Flex* (1) 331

Fig. A.19: Eventos de *Adobe Flex* (2) 332

Índice de tablas

Tabla 3.1: Comparativa de los métodos de Ingeniería Web 67

Tabla 4.1: Comparativa primitivas conceptuales OO-Method/OOWS 88

Tabla 4.2: Comparativa primitivas conceptuales OOWS/OO-Method 89

Tabla 4.3: Plantilla para la especificación de una *Service AIU* 99

Tabla 4.4: Ejemplo de una *Service AIU* 100

Tabla 4.5: Plantilla para la definición del *Summary View AIP* 108

Tabla 4.6: Ejemplo de un *Summary View AIP* 108

Tabla 4.7: Plantilla para la definición de un *Validation Rule AIP* 113

Tabla 4.8: Ejemplo de un *Validation Rule AIP* 114

Tabla 4.9: Ejemplo de un *Argument Setting AIP* 116

Tabla 4.10: Ejemplo de un *Service Navigation AIP* 121

Tabla 4.11: Primitivas seleccionadas para el Modelo de Interacción Abstracto 127

Tabla 5.1: Relaciones de *weaving* con el Modelo de Interacción Abstracto 164

Tabla 6.1: Relación entre los patrones y las webs analizadas 194

Tabla 6.2: Notación de tareas basada en CTT 200

Tabla 6.3: Operadores de la notación CTT 201

Tabla 7.1: Soporte de los escenarios en OOWS y OOWS 2.0 249

Tabla 7.2: Modelo de *weaving* para el contexto *Compare Genes* 256

Tabla 7.3: Diseño del experimento 275

Tabla 7.4: Relación entre las métricas y el cuestionario de evaluación 282

Tabla 7.5: Orden temporal de las tareas 282

Tabla 8.1: Publicaciones realizadas en el marco de la tesis. 297

Capítulo 1

Introducción

1.1 ¿Qué son las aplicaciones Web 2.0?

La Web se ha convertido, por méritos propios, en el medio de comunicación por excelencia del siglo XXI. Resulta difícil decir nada original para justificar el tremendo impacto que tiene, y va a seguir teniendo, en el devenir de nuestra civilización. Entre sus muchas aportaciones, destaca la rapidez con la cual se intercambia la información que, unida a la eliminación de las barreras geográficas, han convertido a Internet en un terreno fértil en el cual las empresas pueden extender sus negocios. Como consecuencia, ha proliferado el número de aplicaciones Web para la resolución de las necesidades de las organizaciones. A diferencia de las aplicaciones tradicionales desarrolladas para una plataforma tecnológica concreta, las aplicaciones Web potencialmente pueden llegar a cualquier tipo de dispositivo. Por lo tanto, este nuevo paradigma de desarrollo se está utilizando, cada vez más, para implementar aplicaciones gubernamentales, de enseñanza a distancia o de gestión empresarial entre otras.

Son varias las ventajas que nos proporciona resolver la problemática de una organización a través de una aplicación Web. En primer lugar, para su utilización el usuario final solo necesita conocer el uso de un navegador Web.

Gracias a este hecho, no es necesario que asimile conocimientos de instalación, configuración y utilización. En segundo lugar, puesto que el desarrollo Web se basa en estándares aceptados (HTTP, HTML, XML, etc.) y tecnologías multiplataforma (*Flash*, *JavaScript*, etc.), se soluciona el problema de generar software para distintos sistemas operativos o dispositivos. La misma aplicación Web puede ser utilizada tanto desde el navegador *Internet Explorer* de *Windows,* como desde un *smartphone* con un navegador móvil. De esta manera, se simplifica, notablemente, el mantenimiento de la aplicación pues siempre se accede desde el mismo servidor y, ante un cambio, no es necesario instalar una nueva versión en todos los dispositivos.

Gracias a las ventajas de este nuevo paradigma de desarrollo, el número de aplicaciones Web ha crecido exponencialmente. Debido a esta rápida evolución y a la complejidad tecnológica añadida, el desarrollo de aplicaciones Web se caracteriza por ser más costoso y complejo que el desarrollo tradicional de software. Además, para agravar dicha situación, la gran mayoría de las aplicaciones Web han sido desarrolladas sin seguir ningún tipo de metodología. Como consecuencia se han detectado graves problemas de mantenimiento, calidad y reusabilidad.

A pesar de dichos problemas, las aplicaciones Web han continuado evolucionando tecnológicamente para abarcar un mayor número de ámbitos. Esta evolución ha sido resumida en un término que ha ganado, rápidamente, una amplia aceptación: Web 2.0. Sin embargo, todavía no existe una definición ampliamente consensuada para el término Web 2.0. Fundamentalmente, porque en esencia la Web 2.0 comparte la misma definición tecnológica que la "Web 1.0". Es decir, se compone de un conjunto de documentos, enlazados mediante hipervínculos, que son accesibles a través de Internet usando el protocolo HTTP.

El origen del concepto Web 2.0 guarda una estrecha relación con la Crisis de las "punto.com" que se produjo a finales de los años 90. Dicha crisis fue producto de la excesiva y artificial revalorización, que tuvieron las empresas de servicios de Internet. Esta crisis puso de manifiesto, que cualquier negocio trasladado a la Web no tenía por qué ser viable, desde el punto de vista económico. También demostró que no bastaba con exponer un producto en la Web para aumentar, instantáneamente, sus ventas. La consecuencia fue la

desaparición, en un corto espacio de tiempo, de numerosos sitios Web dejando perdidas multimillonarias en multitud de inversores. En cierta medida, esta crisis puso en tela de juicio las elevadas expectativas depositadas en la revolución del comercio electrónico. Estaba claro que la Web era útil para hacer negocios y que no estaba desfasada pero, a la misma vez, era evidente que "algo" había fallado. La prueba era la existencia de sitios Web de comercio electrónico que alcanzaron en ese periodo un notable éxito, y que aún perduran, como *Ebay* o *Amazon.*

Después de esta crisis, otras aplicaciones Web, que no estaban directamente relacionadas con el comercio electrónico, adquirieron una elevada repercusión en Internet. El máximo exponente de estas aplicaciones Web es el ecosistema de *Google*, pero también destacan *Wikipedia, YouTube* o, más recientemente, la red social *Facebook*. El indicador de su éxito actual es completamente objetivo: el gigantesco tráfico de usuarios que aglutinan. Además, guardan un característica común que las aleja del modelo de negocio de la "Web 1.0": son gratuitas, en muchos casos, para los usuarios y son ellos su principal activo.

El concepto Web 2.0, tal y como es hoy ampliamente entendido puesto que fue utilizado anteriormente con otros propósitos, es atribuido a *Dale Dougherty* y *Craig Cline* en el proceso de recopilación de ideas para la que sería la primera conferencia sobre la Web 2.0. El objetivo de dicha conferencia era comprender la crisis y poner de manifiesto, la transición que se estaba produciendo en la Web. Cuando fue definido por primera vez el concepto de la Web 2.0 [O'Reilly 2005], se realizó de manera ambigua, porque la definición era una contraposición a los modelos de negocio y las tecnologías provenientes de la "Web 1.0", que habían provocado el fracaso. Como consecuencia, bajo el "paraguas" de la Web 2.0 se han aglutinado una serie de conceptos y tecnologías que, en principio, no guardan relación entre si como la usabilidad de las interfaces, el fomento de la participación de los usuarios o los servicios Web. El denominador común de dichos conceptos y tecnologías es su papel determinante, en el éxito de las aplicaciones Web más relevantes de los últimos años. Sin embargo, aquello que comenzó siendo una simple palabra para destacar a las Webs más populares, se ha convertido en una nueva forma de enfocar tanto el desarrollo Web como los modelos de negocio en Internet [Murugesan 2007].

No es sencillo proporcionar una definición precisa del concepto de aplicación Web 2.0. En esta tesis doctoral definimos el concepto como la suma de las dos facetas o aspectos, destacados en negrita:

a) ***Una faceta colaborativa, en la cual el usuario final es el eje central de la aplicación Web***. En los sitios Web tradicionales, el usuario era un consumidor pasivo de la información que, normalmente, definían los administradores del sitio Web. En la Web 2.0, es el usuario final quien se encarga no solo de crear el contenido del sitio (definiciones en *Wikipedia,* videos en *YouTube*, etc.), sino en valorar qué contenido es de mayor calidad y, en establecer la categorización del mismo. Este cambio del usuario a un rol activo ha propiciado el crecimiento exponencial de contenido en la Web en la forma de: opiniones, noticias y experiencias personales en *blogs*, definiciones de conceptos en los *wikis* o, videos y fotos en páginas de contenido multimedia. Simultáneamente, ha emergido la denominada Web Social, la cual establece una analogía con la Web tradicional enlazando en vez de documentos a usuarios. De esta manera, se han creado, virtualmente, redes sociales en las cuales los usuarios están enlazados entre sí, por las características que les definen en el mundo físico (aficiones, relación laboral, lugares donde estudiaron, etc.). Siendo la participación del usuario esencial, indirectamente, proporcionar una interacción precisa con la aplicación Web se ha convertido en un requisito imprescindible.

b) ***Una faceta tecnológica avanzada, con el objetivo de facilitar la interacción del usuario final con la aplicación Web***. Si analizamos las interfaces de los sitios Web 2.0 más populares veremos que poseen un alto nivel de usabilidad. Para alcanzar dicho nivel, han sido indispensables una serie de tecnologías que han permitido desarrollar interfaces e interacciones más elaboradas. El uso de dichas tecnologías ha dado lugar a las *Rich Internet Applications* o RIA [Duhl 2003], aplicaciones que residen en un servidor Web pero en donde el proceso de la capa de presentación es delegado, parcial o totalmente, al navegador Web cliente. Entre las tecnologías RIA [Noda & Helwig 2005] más destacadas se encuentra: (1) AJAX, para obtener los datos del servidor bajo demanda evitando la recarga completa de una página, (2) *frameworks* de *Javascript*, para añadir lógica de negocio en el lado del cliente y (3) entornos de desarrollo RIA, para implementar interfaces gráficas avanzadas que incluyen animaciones, contenido multimedia e interacciones complejas.

El cumplimiento, en mayor o menor medida, de ambas facetas es lo que otorga el estatus de Web 2.0 a una aplicación accesible desde Internet. De esta forma, *Google Maps* se cataloga como una Web 2.0 como consecuencia de su avanzada interfaz basada en AJAX y *Javascript*, aunque el contenido fundamental de la aplicación, los mapas, no son creados por los usuarios. Por otro parte, el sitio Web 2.0 *Twitter*, herramienta para el denominado *microblogging*, no entraría en el dominio de las RIA. A pesar de ello, la facilidad de interacción con los usuarios convierte a *Twitter,* también, en un claro ejemplo de Web 2.0.

Considerando que una aplicación Web 2.0 no es más que un producto software con unas características concretas, es viable utilizar las principios de la Ingeniería del Software para atacar la problemática de su desarrollo. Lamentablemente, los métodos de Ingeniería del Software tradicionales, no están adaptados para resolver los requisitos específicos de las aplicaciones Web 2.0. La Ingeniería Web [Murugesan, Deshpande *et al.* 2001] surge con el objetivo de aplicar los fundamentos de la Ingeniería del Software sobre el desarrollo sistemático de aplicaciones Web, atendiendo a las características particulares propias de este tipo de aplicaciones. Por lo tanto, se define como una disciplina específica, que estudia los procesos, métodos, técnicas y recursos esenciales para el desarrollo de aplicaciones Web de calidad. Sin embargo, hasta ahora la Ingeniería Web se ha centrado en resolver los retos que ha planteado el desarrollo de aplicaciones "Web 1.0". El objetivo de esta tesis es del analizar como la Ingeniería Web tiene que evolucionar para resolver la problemática específica del desarrollo Web 2.0. En definitiva: determinar cómo tienen que extenderse los métodos de Ingeniería Web actuales, con el objetivo de producir aplicaciones Web 2.0 de calidad.

1.2 Motivación

Aunque el procedimiento para otorgar a una aplicación Web el estatus de 2.0 sea confuso, se observa una clara tendencia a incluir diversas características Web 2.0 en el desarrollo Web. Por ejemplo, aplicaciones Web tradicionales, como *EBay* o *Amazon*, están incorporando tanto interfaces más ricas como una mayor implicación del usuario en su proceso de negocio.

Tanto la Ingeniería Web como el Desarrollo de Software Dirigido por Modelos (DSDM) han proporcionado soluciones al desarrollo de aplicaciones Web. Como consecuencia de la combinación de ambas disciplinas, el entorno académico ha propuesto diversos métodos de Ingeniería Web en los cuales, los modelos conceptuales son artefactos clave en el proceso de desarrollo. Ejemplos de dichos métodos son WebML [Ceri, Fraternali *et al.* 2000], UWE [Koch. 2000], OOH [Meliá & Cachero 2004], OOHDM [Schwabe, Rossi *et al.* 1996] y OOWS [Fons, Pelechano *et al.* 2003] entre otros [Rossi, Pastor *et al.* 2008].

En los últimos años, los métodos de Ingeniería Web han proporcionado resultados interesantes para el desarrollo de aplicaciones Web, donde la recuperación de datos y la navegación a través de la información son las interacciones principales con el usuario. Estos métodos han estado, fundamentalmente, enfocados a los dominios Web que eran más habituales hasta ahora: aplicaciones de comercio electrónico (*e-commerce*) y aplicaciones con recuperación intensiva de información (*data-intensive applications*).

No obstante, el desarrollo de aplicaciones Web ha continuado evolucionando dejando obsoletos dichos métodos en un breve espacio de tiempo. En las aplicaciones Web es cada vez más frecuente encontrar interfaces de carácter multimedia, contenido generado por los usuarios o interacciones complejas, características, todas ellas, que no se encuentran adecuadamente soportadas por los métodos de Ingeniería Web [Preciado, Linaje *et al.* 2005]. Las carencias de estos métodos no se encuentran ni en el análisis de los requisitos, ni en el enfoque dirigido por modelos que proponen, ni en las estrategias para la implementación o la generación de código. Principalmente, las carencias se detectan en el nivel de modelado conceptual ya que los modelos que se utilizan no poseen la suficiente expresividad. En consecuencia, dichos modelos tienen que evolucionar en consonancia a como lo han hecho las propias aplicaciones que representan.

La motivación principal de esta tesis es resolver parte de la problemática relacionada con el desarrollo de aplicaciones Web 2.0 que, actualmente, no se encuentra cubierta por los métodos de Ingeniería Web. Dicha motivación principal es muy ambiciosa, puesto que la Web 2.0 ha introducido diversos aspectos que son candidatos a ser incluidos en dichos métodos. Por nombrar

algunos, la creación de *mashups*, la especificación y explotación de las llamadas redes sociales, la incorporación de contenido multimedia o la definición de las llamadas *folksonomias*. Ante la imposibilidad de abarcarlos todos en el marco de una tesis doctoral, se han seleccionado dos que proporcionan, a nuestro juicio, las mejoras más significativas:

1) **Soporte al modelado avanzado de la interacción:** Tradicionalmente, en los métodos de Ingeniería Web, los modelos principales han sido el modelo de datos o de dominio y el modelo de navegación. Si bien estos modelos continúan siendo relevantes en el ámbito de las aplicaciones Web 2.0, los modelos de interacción que especifican la interfaz entre el usuario y el sistema no han sido definidos con el mismo nivel de detalle. Las interfaces de usuario basadas en tecnologías RIA han puesto de manifiesto las carencias de dichos modelos de interacción. Por un lado, no es posible modelar la gran variedad de componentes gráficos o *widgets* que proporcionan dichas tecnologías. Por otro lado, las interacciones del usuario con los distintos componentes de la interfaz son más elaboradas y de mayor complejidad. Ambos aspectos tienen que ser contemplados, manteniendo la coherencia con los modelos previamente definidos, en el ámbito de la Ingeniería Web.

2) **Fomento de la participación del usuario:** Una de las características fundamentales de una aplicación Web 2.0, es el alto grado de implicación del usuario final para la creación del contenido. La relevancia del usuario final en el marco de una aplicación Web 2.0 es tal, que el éxito o fracaso de la misma depende de conseguir su colaboración. Para conseguir dicho objetivo, se han popularizado una serie de mecanismos que simplifican la interacción con la aplicación. Una consecuencia de la aplicación de estos mecanismos, es la sencillez con la cual se realiza la edición inmediata o la evaluación del contenido presente en el sitio Web. Estos mecanismos de interacción, que se aplican recurrentemente en las aplicaciones Web 2.0, han sido recopilados por diversos autores en forma de patrones de diseño. La gran mayoría de dichos patrones, sí que pueden ser definidos mediante los modelos actuales de los métodos de Ingeniería Web. A pesar de ello, en ningún método de Ingeniería Web se ha definido, explícitamente, un mecanismo para su especificación a nivel de modelado, de tal forma que tienen que ser creados desde cero cada vez que el analista desea utilizarlos.

La incorporación de ambos aspectos en un método de Ingeniería Web dirigido por modelos, es un paso necesario con el objetivo de especificar y generar aplicaciones Web 2.0, que cumplen con dos de los requisitos más demandados en este ámbito [SIIA 2008].

1.3 Objetivos de la tesis

El objetivo general a resolver en esta tesis es el de incorporar las extensiones conceptuales necesarias, para abordar el desarrollo dirigido por modelos de aplicaciones Web 2.0. Para acotar la problemática, esta tesis se centra en dos subobjetivos fundamentales: (1) el modelado avanzado de la interacción incorporando la utilización de tecnologías RIA y, (2) la detección y formalización de patrones de interacción frecuentes en las aplicaciones Web 2.0. Esta tesis tiene que justificar, con precisión, la necesidad de incluir ambos aspectos en los métodos de Ingeniería Web. Las soluciones se deben proponer, además, de manera genérica sin estar ligadas a un método concreto, para aportar una mejora extensible al conjunto de métodos propuestos por la Ingeniería Web. No obstante, a fin de validarlas convenientemente, son introducidas en el método de Ingeniería Web OOWS, con el propósito de obtener un método que pueda ser utilizado para el desarrollo de aplicaciones Web 2.0. Este nuevo método recibe el nombre de OOWS 2.0. Estos objetivos generales nos llevan a formular las siguientes preguntas objeto de investigación:

1. ¿Qué expresividad conceptual tiene que proporcionar un método de Ingeniería Web, para modelar la interacción con el usuario al nivel requerido por las aplicaciones Web 2.0?

2. ¿Cómo pueden ser expresadas en un método de Ingeniería Web dirigido por modelos, las practicas más habituales de la Web 2.0 que fomentan la participación del usuario final?

3. ¿Cuáles son los mecanismos necesarios para introducir las soluciones de las preguntas anteriores, en el marco de un método de Ingeniería Web dirigido por modelos?

Las soluciones propuestas para estas preguntas de investigación se especifican a continuación.

1.4 Solución propuesta

Las tres contribuciones principales de esta tesis responden, respectivamente, a cada una de las tres preguntas de investigación antes planteadas:

1. Para llevar a cabo el proceso de modelado avanzado de la interacción, se ha seguido la separación en niveles de abstracción propuesta por [Calvary, Coutaz et al. 2003]: (1) un nivel abstracto, donde se define un modelo de interacción independiente de la plataforma tecnológica; (2) un nivel concreto, que refina el modelo de interacción abstracto introduciendo información tecnológica sobre la interfaz a desarrollar. Para definir el nivel abstracto, se ha formalizado un Modelo de Interacción Abstracto que mantiene la expresividad conceptual definida en el método OOWS y al cual se le incluye, la expresividad conceptual del Modelo de Presentación OO-Method [Pastor & Molina 2007]. Por otra parte, el nivel concreto ha sido definido utilizando un modelo para la definición de interfaces de usuario mediante tecnologías RIA, denominado Modelo de Interfaz RIA. Para determinar las entidades que forman parte de dicho modelo se ha realizado un análisis de distintas interfaces de usuario pertenecientes a aplicaciones Web 2.0. Como resultado, el metamodelo correspondiente se compone de una serie de entidades abstractas, que sirven de base para crear otros metamodelos específicos de una tecnología RIA a introducir en el método. A modo de ejemplo, se presenta un metamodelo para la tecnología RIA Adobe Flex [Tapper, Labriola et al. 2008].

2. Para formalizar las prácticas más habituales de la Web 2.0 que fomentan la colaboración del usuario, se ha elaborado un catálogo de patrones utilizando modelos conceptuales. Con dicho objetivo se ha realizado un análisis de diversos sitios Web 2.0, para encontrar qué patrones Web 2.0 se aplican con mayor frecuencia. Como resultado

del análisis, se ha propuesto un catálogo compuesto de 14 patrones: *Quick Comment, Tag Definition, Notification, Collaborative Editing, Quick Rating, Reputation, Share Content, Suggestion, Invite, Public Profile, Availability, Ranking, Favorites* y *Subscription*. Además de la descripción textual habitual, la especificación del patrón incorpora un modelo conceptual, que especifica los aspectos de funcionalidad, y un modelo conceptual, que especifica la interacción representada. Estos patrones a nivel de modelado han sido denominados, en la presente tesis doctoral, como patrones Web 2.0.

3. Con el objetivo de validar ambas soluciones, éstas han sido integradas en el método de Ingeniería Web OOWS, definiendo el nuevo método OOWS 2.0. Para incorporar el Modelo de Interfaz RIA en los métodos de Ingeniería Web, se ha seguido una estrategia de integración basada en la definición de un *weaving metamodel* [Fabro, Bézivin *et al.* 2006], que relaciona el metamodelo de un método de Ingeniería Web con el metamodelo de Interfaz RIA propuesto. La instanciación del *weaving metamodel* establece, para cada primitiva conceptual del método, qué componente de Interfaz RIA es utilizado. También, se ha definido una estrategia de integración de los patrones Web 2.0, basada en transformaciones modelo a modelo. Esta estrategia consiste en la definición de una transformación que expresa el patrón Web 2.0, como un modelo perteneciente a un método de Ingeniería Web específico. Como ejemplo de aplicación, se han proporcionado los modelos OOWS 2.0 de un conjunto de patrones. El resultado de ambas integraciones es un método de Ingeniería Web, con la capacidad de modelar aplicaciones Web 2.0 de calidad, que contemplan aspectos avanzados de interacción con el usuario.

1.5 Ámbito de aplicación

El ámbito de aplicación de esta tesis doctoral está claramente acotado al desarrollo de aplicaciones Web 2.0. Dado el amplio abanico de dominios que abarca la Web 2.0, nuestra propuesta es aplicable, principalmente, en el desarrollo de:

- **Web Sociales**: aplicaciones en las cuales el eje central son los usuarios y las relaciones que se producen entre ellos. Ejemplos de este ámbito, son las aplicaciones Web para establecer redes de contactos personales o laborales.

- **Entornos colaborativos**: aplicaciones Web en las cuales la información es creada y revisada por los propios usuarios. Los *wikis* y las webs de clasificación de contenidos (noticias, videos, etc.) forman parte de este ámbito.

- **Catálogos interactivos**: aplicaciones Web en las cuales es fundamental crear un fuerte impacto visual al visitante sobre los productos y servicios ofertados. Los sitios Web de los fabricantes de productos de consumo son un claro ejemplo.

- **Aplicaciones con interacción avanzada**: aplicaciones con la complejidad interactiva de los entornos de escritorio que han sido migradas a la Web. Ejemplos de este ámbito son las herramientas en línea de creación de documentos o de gestión de proyectos.

- **La migración de aplicaciones "Web 1.0":** aplicaciones Web de comercio electrónico o de gestión de la información, que habiéndose desarrollado en el contexto tecnológico de la "Web 1.0", necesitan ser migradas al ámbito de la Web 2.0.

1.6 Estructura de la tesis doctoral

Una vez planteados los objetivos principales, la estructura del resto de la tesis doctoral es la siguiente:

- El **Capítulo 2** presenta los fundamentos metodológicos sobre los cuales se basa la presente tesis doctoral. En este capítulo se aborda la metodología de investigación empleada y se proporciona una visión general, tanto del método OOWS como del método OOWS 2.0.

- El **Capítulo 3** analiza el estado del arte relacionado con la presente tesis. Este estado del arte se ha elaborado desde tres líneas de investigación: trabajos en el ámbito del modelado de la interacción, métodos de Ingeniería Web que han sido extendidos para soportar características de la Web 2.0 y, aproximaciones específicas del desarrollo de aplicaciones Web 2.0.

- El **Capítulo 4** presenta el Modelo de Interacción Abstracto del método OOWS 2.0. Este nuevo modelo emerge a partir de los modelos de OO-Method y OOWS, siendo el núcleo sobre el cual se articula el proceso de modelado del método. Se presentan con detalle las primitivas conceptuales que introduce este nuevo modelo.

- El **Capítulo 5** presenta el Modelo de Interfaz RIA introducido en el método OOWS 2.0. En este capítulo, se justifica, en primer lugar, la necesidad de este nuevo modelo. A continuación, se especifican tanto las primitivas conceptuales que lo componen, como una estrategia de integración en el marco de un método de Ingeniería Web.

- El **Capitulo 6** introduce el concepto de patrón Web 2.0. En este capítulo, se realiza un análisis de un conjunto de aplicaciones Web 2.0 para determinar un catálogo de patrones Web 2.0. Utilizando de los patrones seleccionados, se detalla cómo representarlos a través de modelos conceptuales. Por último, se describe una estrategia de integración, con los métodos de Ingeniería Web, basada en transformaciones modelo a modelo.

- El **Capítulo 7** presenta las líneas generales para llevar a cabo una evaluación de la viabilidad del método OOWS 2.0. Esta evaluación se articula a través de una demostración de laboratorio basada en una aplicación Web 2.0. Utilizando esta aplicación se han realizado tres tareas: (1) un análisis de la mejora de la expresividad conceptual de OOWS 2.0 con respecto a OOWS, en el marco de la aplicación seleccionada; (2) el modelado mediante OOWS 2.0 de dos escenarios de la aplicación; (3) un diseño experimental para evaluar las mejoras de OOWS 2.0 sobre el método OOWS.

- El **Capítulo 8** presenta las conclusiones de la presente tesis doctoral, las líneas de trabajo futuro y las publicaciones académicas realizadas.

Capítulo 2

Fundamentos Metodológicos

En el presente capítulo se detallan los distintos aspectos metodológicos en los cuales se fundamenta la presente tesis doctoral. En primer lugar, se describe la metodología de investigación empleada en la sección 2.1. A continuación, en la sección 2.2, se proporciona una visión general del método OOWS, en el cual se basa la presente tesis, y del método OOWS 2.0, contribución principal de la tesis doctoral. En particular, se detallan las distintas fases y los modelos involucrados en ambos métodos. En la sección 2.3, se describe el Modelo de Presentación de OO-Method ya que es una pieza clave para la especificación del Modelo de Interacción Abstracto propuesto. En la sección 2.4, se introducen los Modelos de Navegación y Presentación del método OOWS, para su posterior comparación con las contribuciones de la presente tesis. Por último en el sección 2.5, se exponen las conclusiones.

2.1 Metodología de investigación: Design Science

La hipótesis que esta tesis busca confirmar o refutar es si es viable extender los métodos de Ingeniería Web dirigidos por modelos para abordar el desarrollo de aplicaciones Web 2.0.

En esta tesis se ha utilizado como metodología de investigación el paradigma denominado *Design Science* [March & Smith 1995]. Este paradigma propone que la investigación científica se lleve a cabo mediante la creación de artefactos que resulten innovadores en su campo. De esta forma, tanto el conocimiento y comprensión del problema a resolver como su solución, se obtienen a través de la construcción y posterior aplicación de dichos artefactos. En concreto, en esta tesis doctoral, nos hemos basado en el *framework* conceptual para la investigación en Sistemas de Información (SI) propuesto por [Hevner, March et al. 2004]. Dicho *framework* (ver Fig. 2.1) se utiliza para definir las distintas etapas de una investigación basada en *Design Science*. El *framework* se compone de tres elementos fundamentales:

- El **entorno** (*environment*) define el espacio del problema en el cual se encuentra el objeto de interés de la investigación. En el dominio de la investigación en SI, dicho entorno se compone de las personas, las organizaciones y las tecnologías SI involucradas. Existe una clara relación entre estos tres elementos porque los objetivos, tareas y problemas que definen las necesidades del negocio son percibidos por las personas de la organización, en base a las tecnologías utilizadas para llevarlos a cabo. Todos estos elementos conforman el problema tal y como es percibido por el investigador. Siguiendo este procedimiento se obtiene una investigación relevante, ya que su objetivo fundamental es el desarrollo de soluciones tecnológicas que tratan problemas reales de una organización.

- El siguiente componente es la **base de conocimiento** (*knowledge base*), la cual proporciona las metodologías y los fundamentos utilizados para llevar a cabo la investigación. Por un lado, los resultados previos en investigación sobre SI proporcionan los fundamentos de la investigación, mientras que las metodologías son utilizadas para justificar y/o evaluar la investigación. El rigor se consigue, por lo tanto, seleccionando los fundamentos y metodologías que resuelven de forma adecuada el problema objeto de investigación.

- Una vez definido el entorno y establecida la base de conocimiento, el siguiente objetivo es el de crear y evaluar un **artefacto** (*artifact*) que

cumpla las necesidades de negocio. En el contexto de la *Design Science*, un artefacto se define como una herramienta, método, modelo o instanciación de otra solución previa que sea útil para resolver las necesidades detectadas. Por lo tanto, un artefacto que resuelva problema inexistentes (no definidos en el entorno) carece de sentido.

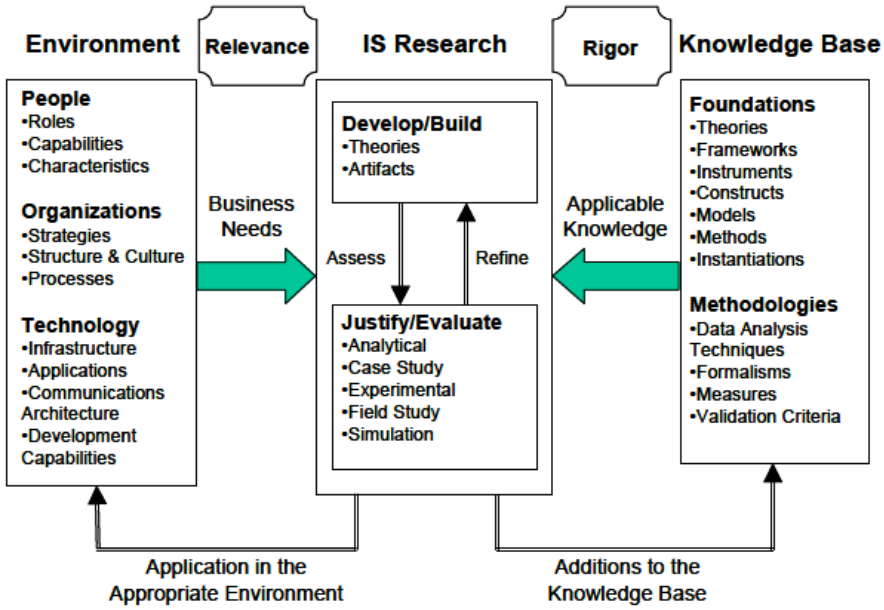


Fig. 2.1: *Framework* de investigación utilizado

Como última etapa de la metodología, las contribuciones de la investigación son aplicadas a las necesidades del negocio en un entorno adecuado y además, son añadidas a la base de conocimiento. Conforme los resultados son añadidos a la base de conocimiento, estos se convierten en prácticas recomendadas para llevar a cabo investigaciones futuras. De esta forma, la construcción de un SI se aborda mediante la aplicación a problemas conocidos, de los elementos presentes en la base de conocimiento.

En el contexto de esta tesis, la metodología *Design Science* se ha aplicado tal y como se describe a continuación. En primer lugar, nuestro entorno viene definido por las organizaciones que desean incorporar características de la

Web 2.0 a sus SI. Tal y como muestran distintos informes [Bughin & Manyika 2007; SIIA 2008], alrededor del 75% de los responsables de SI encuestados han incorporado o piensan incorporar en el futuro cercano características de la Web 2.0 en su organización. Las personas que demandan dichas características son, principalmente, el personal de marketing para fomentar la fidelidad del cliente con la organización. La necesidad de negocio surge como consecuencia de la inexistencia en la actualidad de herramientas de desarrollo. Dicho análisis del entorno es el que ha establecido la motivación de la presente tesis doctoral: proporcionar un método de Ingeniería Web que soporte las características de la Web 2.0 requeridas por las organizaciones.

Para afrontar dicha necesidad del entorno, nuestra base de conocimiento se compone de distintos artefactos propuestos por la Ingeniería Web, la Ingeniería del Software y la comunidad IPO (Interacción Persona-Ordenador). Fundamentalmente, los artefactos seleccionados son métodos de Ingeniería Web y modelos conceptuales que tratan el modelado de IU en ambientes Web. Para la construcción de la base de conocimiento, se ha realizado un estudio sobre los distintos trabajos que han abordado, mediante modelos conceptuales, alguno de los problemas planteados en la presente tesis. Dicho estudio ha permitido detectar las carencias en la base de conocimiento. En concreto, se han encontrado varias aproximaciones que abordan de forma parcial el desarrollo de IU para tecnología RIA, y extensiones a nivel de modelado de los métodos de Ingeniería Web actuales. Sin embargo, pocos trabajos tratan la faceta social o colaborativa de la Web 2.0

Para resolver el problema de entorno planteado, los artefactos que se han desarrollado han sido los siguientes:

- Un metamodelo de interacción abstracto que captura y extiende la expresividad conceptual actual de los métodos OOWS y OO-Method.

- Un metamodelo para la especificación de IU utilizando tecnologías RIA.

- Un catálogo de patrones conceptuales que representan las prácticas de interacción habituales en las aplicaciones Web 2.0.

- Un método de Ingeniería Web dirigido por modelos denominado OOWS 2.0 que incluye los tres artefactos anteriores.

Dichos artefactos se han definido en consonancia con la base de conocimiento, es decir, con los métodos de la Ingeniería Web y de la Ingeniería del Software y, en especial, con las tecnologías estándar para el desarrollo de modelos conceptuales (MOF y UML).

Para la validación de cada uno de los artefactos se ha seguido una aproximación diferente en cada caso. En primer lugar, para el metamodelo de interacción abstracto no se ha realizado una validación explicita, puesto que se compone de primitivas conceptuales previamente validadas en los métodos OOWS y OO-Method.

Para comprobar si el metamodelo RIA es lo suficientemente expresivo se ha seguido una validación basada en escenarios. Para dicha validación, se ha seleccionado una interfaz de usuario proveniente de una aplicación Web 2.0 real. Para dicho especificar dicho escenario se han creado los modelos de interfaz correspondientes.

Para definir el catálogo de patrones, se han analizado distintas aplicaciones Web 2.0 y se han extraído las soluciones que enfatizan la colaboración con el usuario final. A continuación, se ha realizado un estudio para determinar su grado de aplicación en un conjunto de aplicaciones Web 2.0. Las soluciones que son adoptadas en un mayor porcentaje han sido definidas como patrones conceptuales e incluidas en el catálogo.

Por último, se ha extendido el método OOWS para validar la estrategia de integración de dichos artefactos y llevar a cabo su aplicación. De esta manera se ha definido el método OOWS 2.0 para el desarrollo de aplicaciones Web 2.0. Este método ha sido comparado con el método OOWS para distinguir la mejora en la expresividad conceptual de los nuevos modelos introducidos. Como paso previo a su implementación industrial, se ha realizado un estudio de la viabilidad del método OOWS 2.0.

2.2 Visión general del método OOWS y del método OOWS 2.0

El método OOWS (*Object-oriented Web Solutions*) es un método dirigido por modelos para el desarrollo de Aplicaciones Web definido por [Fons 2008]. OOWS también se define como una extensión metodológica del método OO-Method para tratar la problemática específica del desarrollo de este tipo de aplicaciones. El método como tal define una serie de fases para capturar la expresividad necesaria para especificar una aplicación Web. La principal ventaja de utilizar una aproximación dirigida por modelos es que el método no propone una fase de implementación, sino en su lugar define un proceso de generación de código a partir de dichos modelos.

Para comprender mejor las distintas fases y artefactos que componen el método OOWS se ha definido el metamodelo del método. Con dicho fin se ha utilizado la técnica propuesta por [Weerd & Brinkkemper 2009] proveniente de la Ingeniería de Métodos (*Method Engineering*). Esta técnica, que se basa a su vez en la propuesta de [Saeki 2003], utiliza una combinación de dos diagramas UML (ver Fig. 2.2):

- En la parte izquierda del modelo, se utiliza una adaptación del diagrama de actividades de UML para especificar el proceso del método. Este diagrama describe las actividades y sub-actividades del método junto a las transiciones entre ellas.

- En la parte derecha del modelo, se utiliza una adaptación del diagrama de clases de UML que modela los artefactos, que se utilizan en cada una de las actividades del método, y las relaciones entre ellos. De esta forma, se captura una vista de los distintos artefactos que se producen.

Ambos diagramas son integrados en único modelo y relacionados entre sí. Las actividades se enlazan con los distintos artefactos, mediante flechas punteadas, para indicar que en esa actividad específica se genera o utiliza el artefacto correspondiente. El modelo resultante es denominado *Process-Deliverable Diagram* (PDD) donde cada fase se representa como un fragmento metodológico (*method fragment*) acorde con la terminología de la Ingeniería de Métodos.

La Fig. 2.2 muestra el PDD principal del método OOWS. En este modelo se observa que el método se compone de tres fases principales y una opcional:

1. **Requeriments Modelling**: el método OOWS define un técnica específica para la captura de requisitos para aplicaciones Web. Esta técnica propuesta por [Valderas, Fons et al. 2005] se basa en la construcción de un modelo a partir de las necesidades del cliente y la misión de la aplicación. Este modelo denominado *Task-based Requirements Model* describe los requisitos mediante un árbol de tareas que sigue la notación *ConcurTaskTree* (para más detalles sobre esta notación consultar la sección 6.2). Cada requisito es asociado a una tarea la cual incluye además, una descripción detallada de las entidades que participan en la misma. La propuesta también presenta un proceso de transformación basado en gramáticas de grafos para obtener, a partir de este modelo de tareas, una versión preliminar de los modelos OOWS de la aplicación (OOWS Skeleton Model).

2. **Conceptual Modelling:** esta es la fase principal del método en la cual se aborda la especificación de los modelos conceptuales de la aplicación Web. Por su relevancia es descrita con mayor detalle a continuación.

3. **Models Compilation:** una vez definidos los modelos, éstos son utilizados como entrada del compilador de modelos OOWS. Definimos el compilador de Modelos OOWS como un conjunto de reglas de transformación, definidas mediante el lenguaje XPand [OAW 2008], que transforman los modelos a código ejecutable. Estas reglas generan a partir de una especificación OOWS, una interfaz Web y la funcionalidad necesaria para integrarse con la lógica de negocio definida mediante OO-Method. El código generado se basa en el framework WIF, implementado en PHP y descrito en [Valverde, Valderas *et al.* 2007], el cual también aborda la integración con la funcionalidad generada con la herramienta *OLIVANOVA* [CARE 2010]. El resultado de esta actividad es un prototipo funcional (*Application Prototype*) de la aplicación Web.

4. **Presentation Design:** esta actividad opcional del método consiste en la definición de un conjunto de plantillas CSS (*Stylesheet*) para especificar los aspectos visuales (colores, tipografía, etc.) de la aplicación Web. Cada plantilla se basa en un conjunto de etiquetas predefinidas que asocian una regla de estilo específica a un elemento de la aplicación Web generada. En función de las etiquetas utilizadas, se distinguen dos tipos de plantillas: (1) independientes del dominio, si las reglas de estilo se asocian a las primitivas conceptuales de OOWS/OO-Method o a los términos que introduce el *framework*; (2) dependientes del dominio, si las reglas de estilo se asocian a conceptos propios del dominio de la aplicación descritos mediante el Modelo de Objetos.

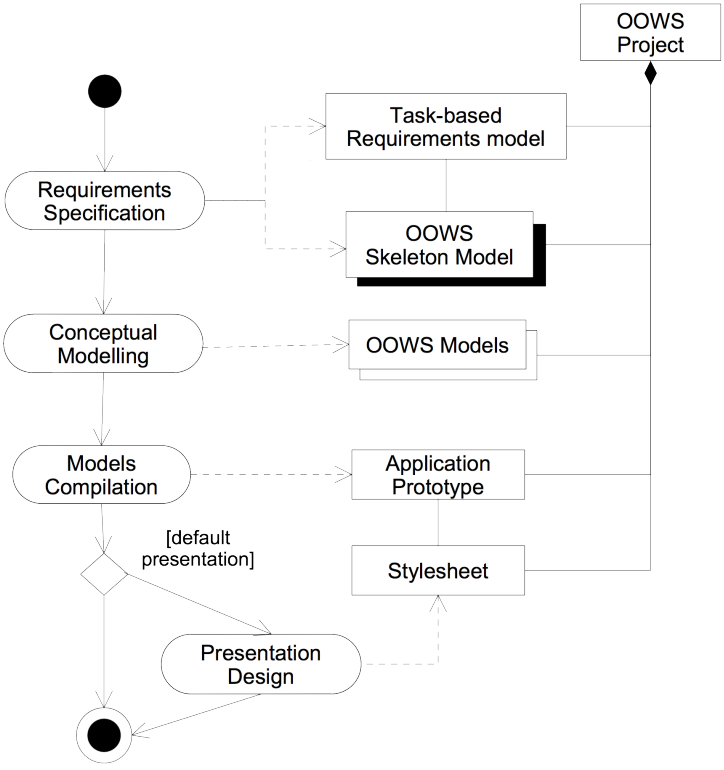


Fig. 2.2: Metamodelo principal del método OOWS

La fase de modelado conceptual del método OOWS es, a efectos prácticos, donde reside la expresividad del método. En esta fase se crea un modelo conceptual que describe una aplicación Web. El PDD de la Fig. 2.3 describe esta fase compuesta de cuatro actividades:

1. **OO-Method Modelling:** la primera actividad consiste en la construcción de los modelos de OO-Method: el Modelo de Objetos, el Modelo Funcional y el Modelo Dinámico. Estos tres modelos provienen del método OO-Method y comparten la misma semántica. La función de estos modelos es la de representar la información y la funcionalidad de la aplicación describiendo el conjunto de clases, y como el estado de sus objetos cambia a través de la ocurrencia de servicios. Desde el punto de vista del modelado, el método OOWS solo hace uso del Modelo de Objetos si bien, los otros dos modelos, son fundamentales a la hora de generar la funcionalidad del sistema.

2. **User Modelling:** la siguiente actividad es la construcción del Diagrama de Usuarios que describe a los usuarios que participan en la aplicación. Este modelo se relaciona con aquellas clases del Modelo de Objetos OO-Method que han sido definidas como agentes. Este diagrama se describe con detalle en la sección 2.4.

3. **Navigation Modelling:** el modelado de la navegación describe cómo los distintos usuarios acceden al sistema. El conjunto de navegaciones definidas para un usuario conforma su Mapa de Navegación (*Navigational Map*). Un Mapa de Navegación se compone de un conjunto de Contextos de Navegación en los cuales se describe la información y la funcionalidad ofertadas. Este modelo se describe con más detalle en la sección 2.4.

4. **Presentation Modelling:** en esa actividad se define un modelo para determinar la presentación de la información disponible en los distintos Contextos de Navegación. Esta presentación se describe mediante un conjunto de patrones que definen el modelo. Estos patrones se introducen con más detalle en la sección 2.4.

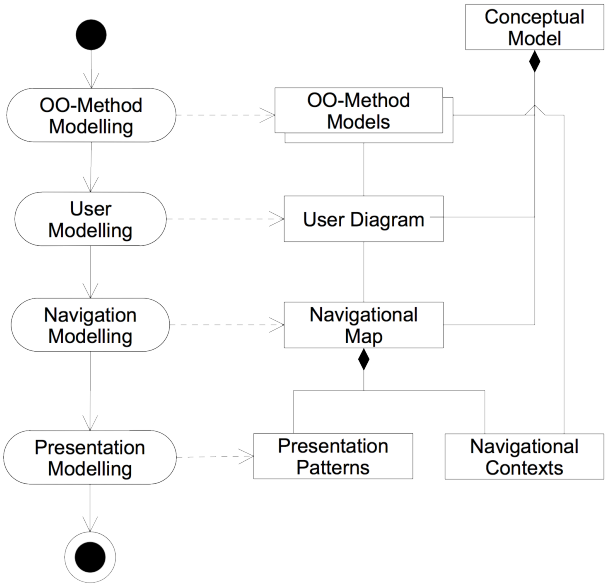


Fig. 2.3: Fase de modelado conceptual en el método OOWS

El resultado final de esta fase del método OOWS es un modelo conceptual que engloba los modelos descritos en las distintas actividades. Este modelo conceptual es utilizado como entrada en la fase de generación de código.

El método OOWS 2.0 es una extensión del método OOWS, fundamentalmente, a nivel conceptual tal y como se justifica a lo largo de la tesis. Por dicha razón, las fases principales del método descritas en la Fig. 2.2 se mantienen. La variación principal se produce en la fase de modelado conceptual en donde se introducen una serie de modelos para abordar, de forma más efectiva, el modelado de aplicaciones Web 2.0. Este conjunto de modelos conforman las contribuciones de la presente tesis y son descritos en detalle en los capítulos correspondientes. El PDD, que describe la fase de modelado conceptual del método OOWS, se muestra en la Fig. 2.4 y se compone de dos actividades obligatorias y dos actividades opcionales:

1. **OO-Method Modelling:** esta actividad es idéntica a la descrita en el caso del método OOWS. Por lo tanto, al igual que el método anterior, el método OOWS 2.0 también reutiliza los modelos definidos en OO-Method.

2. **Interaction Modelling:** en esta actividad se construye un modelo que representa la interacción entre el usuario y el sistema: el Modelo de Interacción Abstracto. Este modelo sustituye al Modelo de Navegación de OOWS ya que no describe únicamente la navegación, sino que se centra en los aspectos de la interacción con el sistema. Sin embargo, reutiliza de OOWS los conceptos de "mapa" y "contexto" puesto que se compone de un Mapa de Interacción descrito en base a un conjunto de contextos de interacción. Este modelo es descrito en el capítulo 4.

3. **Web 2.0 Patterns Modelling:** en esta actividad el analista aplica, opcionalmente, un conjunto de patrones Web 2.0. Los patrones Web 2.0 son representaciones conceptual de problemas habituales que se producen en el desarrollo de aplicaciones Web 2.0. Estos patrones son transformados en una representación del Modelo de Interacción Abstracto, que soluciona la problemática descrita por el patrón. El conjunto de patrones Web 2.0 disponibles y su mecanismo de aplicación son descritos en el capítulo 6.

4. **RIA Interface Modelling:** en esta actividad opcional el analista realiza un modelo de la interfaz de la aplicación. En el método OOWS esta actividad no esta contemplada ya que a partir del Modelo de Navegación se genera una interfaz por defecto. En el método OOWS 2.0 se define, de forma explícita, una actividad de modelado de una interfaz basada en una tecnología RIA. La realización de esta actividad es opcional porque el método genera una versión por defecto del Modelo de Interfaz RIA. La especificación de esta interfaz se construye en base a dos modelos interrelacionados: un Modelo de Interfaz RIA, que describe los *widgets* y el comportamiento ante los eventos de interfaz, y un modelo de *weaving*, que relaciona los distintos *widgets* con las primitivas conceptuales del Modelo de Interacción Abstracto. Ambos modelos son descritos con detalle en el capítulo 5.

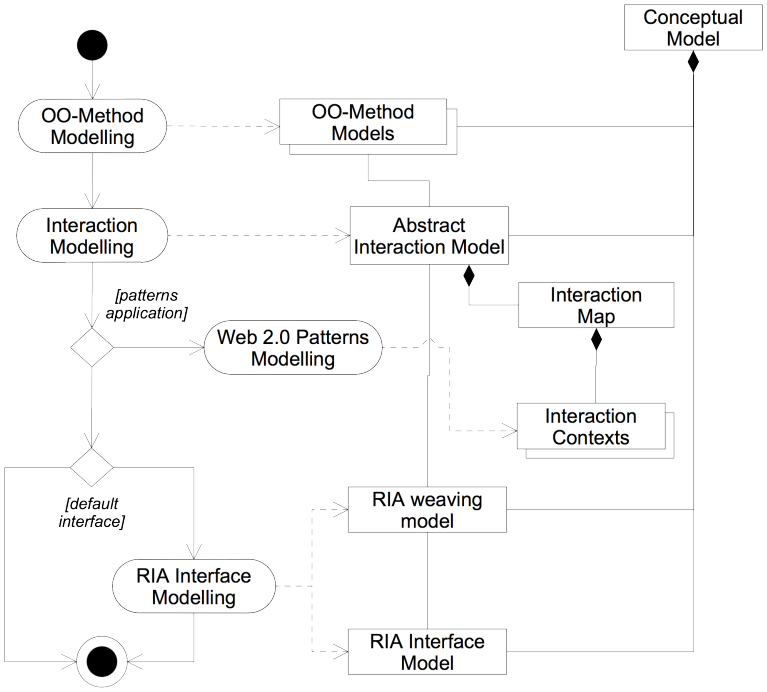


Fig. 2.4: Fase de modelado conceptual del método OOWS 2.0

Al igual que en el método OOWS, el resultado de esta actividad es un modelo conceptual pero en este caso compuesto de: un conjunto de modelos OO-Method, un modelo de interacción abstracto, un modelo de *weaving* y un Modelo de Interfaz RIA. La Fig. 2.5 muestra los modelos conceptuales que componen ambos métodos. En el caso de OOWS 2.0, este conjunto de modelos es utilizado en la fase de generación de código para la producción de una aplicación Web 2.0 completamente funcional. Esta fase de generación difiere de la presentada en el método OOWS sin embargo, al quedar fuera del alcance de la presente tesis doctoral, no es abordada aquí.

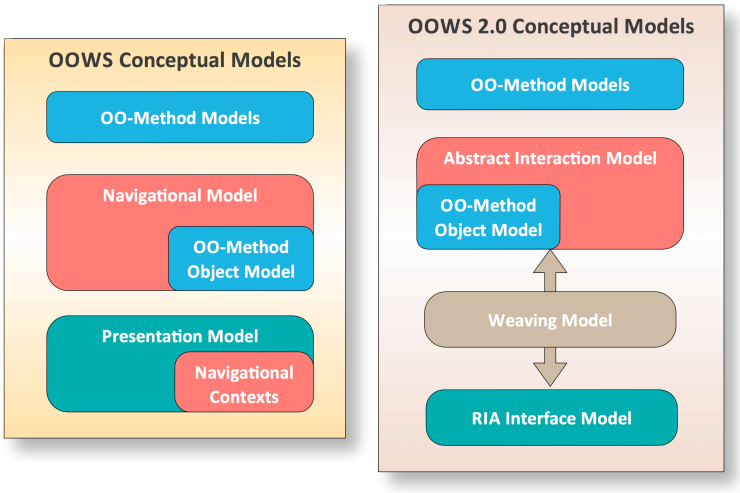


Fig. 2.5: Modelos Conceptuales de OOWS y OOWS 2.0

2.3 El Modelo de Presentación de OO-Method

El Modelo de Presentación de OO-Method [Molina 2003] es un modelo para la construcción de IU orientadas a objetos. Este modelo se sustenta sobre el resto de modelos OO-Method (Modelo de Objetos, dinámico y funcional) para obtener las entidades del dominio relevantes a la hora de construir la interfaz. De esta manera, se asocia la interfaz con objetos del dominio, para mostrar su información, o con servicios de las clases, para proveer funcionalidad. Los constructores básicos del modelo son un conjunto de patrones conceptuales que abstraen tanto una interacción genérica como la IU para llevarla a cabo. El modelo se define en base a patrones porque éstos ocultan al analista gran parte de la complejidad implícita de la interfaz, y permiten crear los modelos más fácilmente. El lenguaje de patrones en el cual se basa el Modelo de Presentación se denomina Just-UI [Molina, Melia *et al.* 2002].

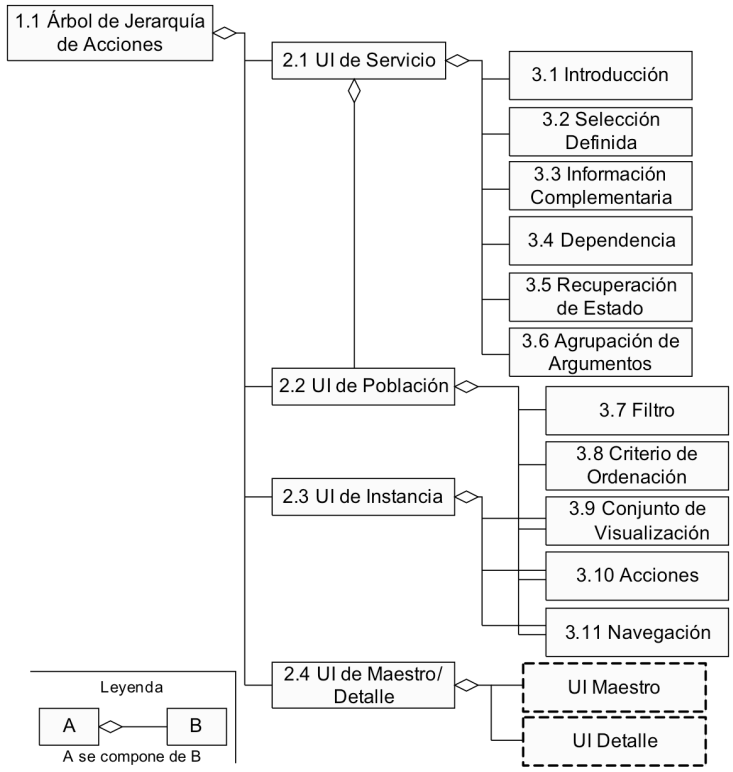


Fig. 2.6: Estructura del Modelo de Presentación OO-Method

El Modelo de Presentación se estructura en una jerarquía de tres niveles tal y como se muestra en la Fig. 2.6. A continuación se detallan las características de cada uno de estos niveles

Nivel 1

En este nivel únicamente se encuentra el patrón conceptual de Árbol de Jerarquía de Acciones (AJA) o *Hierarchical Action Tree* (H.A.T). Este patrón organiza el acceso a la funcionalidad por parte del usuario a través de una abstracción en forma de árbol. De esta manera, cada usuario qué accede al sistema tiene asociado un árbol compuesto del conjunto de patrones sobre los cuales tiene visibilidad.

Nivel 2

Este nivel está compuesto por las Unidades de Interacción (UI) o *Interaction Unit*. Una UI es una unidad de presentación que abstrae tanto la presentación visual de la interfaz, es decir, qué componentes gráficos van a ser utilizados como el comportamiento asociado, es decir, la comunicación entre el usuario y la propia UI. El Modelo de Presentación propone cuatro tipos de UI: (1) UI de Servicio (*Service)*: representa un formulario para que el usuario introduzca los argumentos necesarios en la ejecución de un servicio; (2) UI de Instancia (*Instance*): muestra la información asociada al estado actual de un objeto o, en otras palabras, el valor de sus atributos; (3) UI de Población (*Population*): muestra mediante un listado tabular el estado actual del conjunto de objetos pertenecientes a una clase; (4) UI Maestro-Detalle (*Master-Detail*): es una UI compuesta de dos UI, ya sean de Población o Instancia, definidas sobre sendas clases que se encuentran relacionadas estructuralmente. De esta forma, una UI aporta la información maestra mientras que la otra muestra la información de detalle. En consecuencia, cuando el usuario selecciona un objeto de la UI Maestra, se muestran todos los objetos relacionados con el objeto seleccionado en la UI Detalle.

Nivel 3

En el último nivel nos encontramos los Patrones Elementales (PE). Estos patrones restringen y precisan el comportamiento de las diferentes UI del nivel anterior, a fin de proporcionar una mayor expresividad. Estos patrones son:

- *Entry* (Introducción): restringe el conjunto de valores de entrada que pueden ser introducidos al sistema filtrando valores incorrectos, o guiando al usuario en su introducción mediante máscaras de edición y mensajes de ayuda.

- *Defined Selection* (Selección definida): proporciona un conjunto de valores predefinidos válidos que el usuario puede seleccionar.

- *Complementary Information* (Información complementaria): muestra información adicional sobre una instancia con el fin de ayudar al usuario.

- *Argument Dependency* (Dependencia): establece una relación de dependencia entre dos datos de entrada de tal forma que cuando uno de ellos es introducido, el otro cambia su valor en función de una fórmula lógica.

- *State Recovery* (Recuperación de estado): inicializa el valor un conjunto de datos en función del valor de los atributos de un objeto que es recuperado previamente.

- *Argument Grouping* (Agrupación de argumentos): conforme a un criterio definido, agrupa un conjunto de datos para simplificar su introducción por parte del usuario.

- *Filter* (Filtro): en una UI de Población, permite al usuario establecer una condición de búsqueda para obtener un conjunto acotado de información.

- *Order Criteria* (Criterio de ordenación): ordena los objetos recuperados mediante una UI de Población en función del valor de un atributo.

- *Display Set* (Conjunto de visualización): define el conjunto de atributos que se muestran en una UI de Población.

- *Actions* (Acciones): determina el conjunto de servicios que un usuario puede ejecutar en una UI determinada.

- *Navigations* (Navegación): muestra en una nueva UI, un conjunto de información relacionado con el objeto seleccionado por el usuario.

Además de los patrones elementales originales propuestos por [Molina, Melia *et al.* 2002], recientemente se han introducido un conjunto de patrones adicionales que forman parte de la herramienta industrial *OLIVANOVA*. Tal y como se documenta en [CARE 2009] estos patrones son:

- *Editable Display Set* (Conjunto de visualización editable)*:* permite modificar el valor de los atributos de las instancias que se muestran en una población y, posteriormente, almacenar los cambios.

- *Conditional Navigation* (Navegación condicional)*:* define una navegación hacia una UI como consecuencia de la ejecución de un servicio, siempre y cuando, se cumpla una fórmula condicional.

- *Navigational Filtering* (Filtrado navegacional)*:* filtra la población de una clase como consecuencia de una navegación hacia una UI destino.

- *Population Preload* (Precarga): indica que la población tiene que ser cargada por defecto cuando se muestra una UI.

- *Tree view* (Disposición en árbol)*:* representa una UI Maestro-Detalle como un árbol desplegable cuyos nodos son la información recuperada.

- *Outbound Arguments* (Visualización de resultados): permite mostrar al usuario un conjunto de argumentos como resultado de la ejecución de un servicio.

La herramienta *OLIVANOVA* proporciona un editor visual para crear el Modelo de Presentación. Este editor se basa en un árbol cuyos primeros nodos son las clases que forman el esquema conceptual. Para cada clase se define qué UI utiliza la información de dicha clase. A continuación, a partir de cada UI, se definen los patrones elementales que son utilizados para restringir su comportamiento. En la Fig. 2.7 se muestra una captura de dicho editor. Se observa la definición de una UI de Población para la clase Contador, que tiene asociados los patrones elementales "Conjunto de visualización", "Criterio de ordenación" y "Filtro".

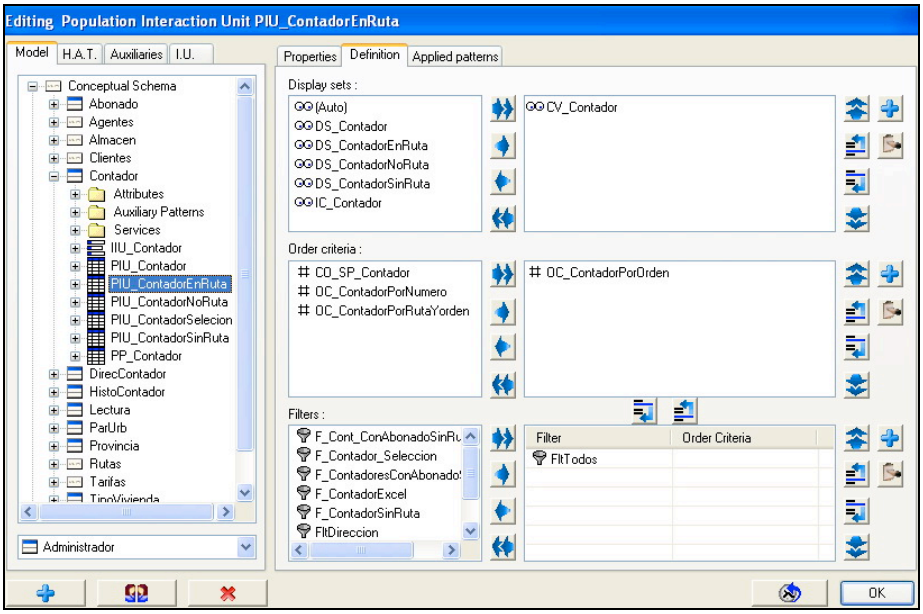


Fig. 2.7: Editor del Modelo de Presentación en *OLIVANOVA*

2.4 Los Modelos de Navegación y de Presentación de OOWS

El principal mecanismo conceptual del método OOWS es el llamado Modelo de Navegación cuya misión es describir qué información del sistema es mostrada al usuario, la funcionalidad del mismo que puede ejecutar y las vías disponibles, para acceder a dicha información/funcionalidad. En OOWS, al igual que en otros métodos que siguen una aproximación hipermedia, una aplicación Web se estructura como una red de nodos que se encuentran enlazados entre sí formando un grafo dirigido. De esta forma, el usuario en un momento determinado se encuentra en un nodo con el cual interactúa.

En el Modelo de Navegación, los distintos tipos de usuarios se organizan mediante un Diagrama de Usuarios que muestra su rol en el sistema, su accesibilidad al mismo y las relaciones existentes entre ellos. El Modelo de Navegación se compone de un conjunto de Mapas de Navegación, que represen-

tan y estructuran la visión global del sistema para cada tipo de usuario definiendo su navegación permitida. El Mapa de Navegación se representa directamente usando un grafo dirigido en el cual los nodos son los Contextos de Navegación y los arcos son los enlaces (o vínculos) de navegación tal y como muestra la Fig. 2.8.

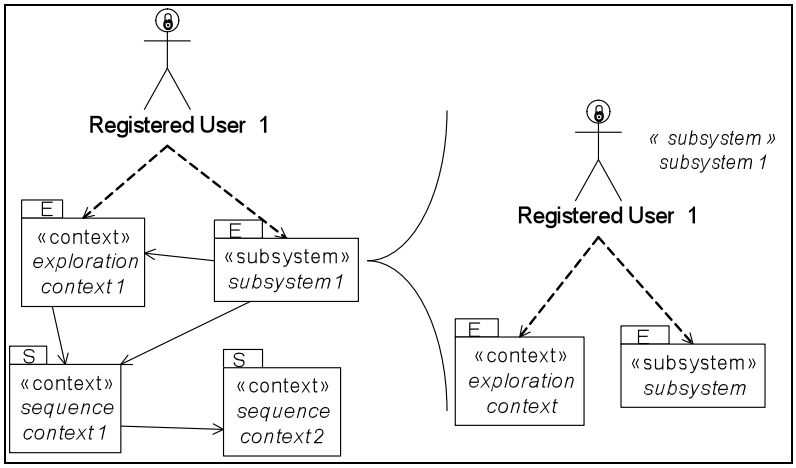


Fig. 2.8: Ejemplo de Mapa de Navegación OOWS

Un Contexto de Navegación está formado por Unidades de Interacción Abstractas o *Abstract Information Units* (AIU) cada una de las cuales, representa una vista sobre un conjunto de datos y/o servicios. Son unidades porque constituyen el elemento lógico básico para la definición de los Contextos de Navegación. De interacción porque representan una acción/respuesta por parte del usuario, una navegación o la activación de un servicio. Y por último son abstractas, porque solo se especifican qué datos y/o servicios se visualizarán en el contexto, pero no como se presentarán.

Cada AIU (ver Fig. 2.9) está compuesta por un conjunto de clases navegacionales, estereotipadas con la palabra reservada «*view*», que hacen referencia a clases identificadas en el Modelo de Objetos. Con estas clases se define la visibilidad ofertada al usuario en un nodo, tanto de los atributos de la clase como de los servicios que puede activar. Toda AIU tiene obligatoriamente una clase navegacional principal llamada Clase Directora (*Manager Class)* y, opcionalmente, otras que contribuyen a complementar la informa-

ción de esta clase llamadas Clases Complementarias (*Complementary Classes*). Las clases navegacionales están unidas entre sí por relaciones binarias unidireccionales, que son definidas sobre una relación de agregación o de herencia existente entre las dos clases en el Modelo de Objetos.

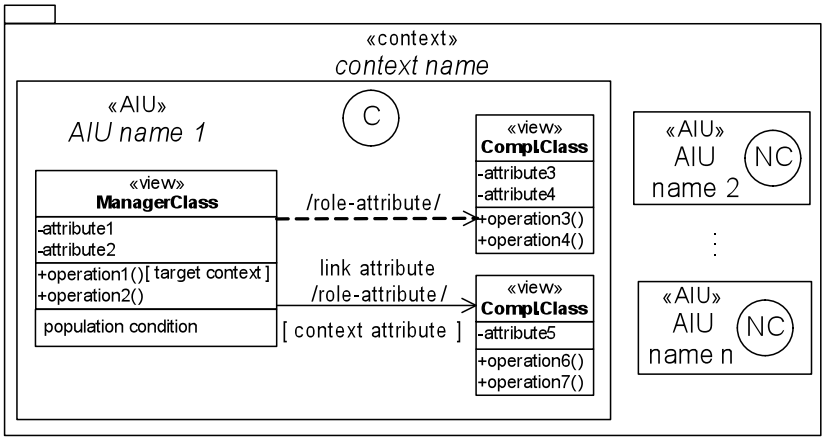


Fig. 2.9: Ejemplo de la definición de una AIU en OOWS

En el Modelo de Navegación, se definen dos tipos de relaciones entre clases navegacionales: (1) una relación de dependencia de contexto (se representa gráficamente mediante una flecha discontinua), que indica una recuperación de información relacionada de las instancias de la clase complementaria a través de su relación en la AIU y (2) una relación de contexto (se representa gráficamente con una flecha continua) que define, además, una navegación a un nodo navegacional destino causando la aparición de un vínculo de navegación en el Mapa de Navegación. Para que dicha navegación sea posible, la Clase Directora del contexto destino tiene que ser la misma que la clase complementaria sobre la que se define la relación de contexto. Los servicios pueden incluir enlaces de servicio para indicar el contexto destino que se alcanzará después de la ejecución del servicio.

Además de las interacciones básicas para la recuperación de información y la ejecución de la funcionalidad, OOWS dispone de mecanismos adicionales para la definición de interacciones más complejas. Estos mecanismos son:

- **Índice**: proporciona un acceso indexado a la información de la clase directora de una AIU. Cuando un Índice es activado, se crea el conjunto de valores posibles que puede tener un atributo de la clase directora. Al seleccionar uno de estos valores por parte del usuario, se visualizan únicamente las instancias que posean dicho valor.

- **Filtro:** restringe el conjunto de objetos recuperados de la clase directora en función de una fórmula condicional basada sobre algún atributo de esta clase. En OOWS, se definen dos tipos de filtros dependiendo del tipo de fórmula utilizada: (1) dinámico, si el filtro es definido mediante una formula abierta, es decir, el valor del atributo sobre el cual se establece la condición de filtrado no es especificado en la fórmula o (2) estático, si el filtro se define como una fórmula cerrada. En el primer caso, es el usuario quien completará la fórmula en tiempo de ejecución introduciendo un valor.

- **Vista de resultados:** este mecanismo se aplica opcionalmente cuando se utiliza un Filtro o un Índice. Define una vista de información parcial del conjunto de instancias recuperadas mediante el Filtro o el Índice. De esta forma, la información se muestra al usuario de una forma más compacta.

- **Operación de sesión:** mecanismo que permite la ejecución de un conjunto de funcionalidad predefinida, cuando el usuario inicia o termina su sesión con el sistema.

- **Usuario conectado**: este mecanismo permite acceder a la información del usuario conectado a la aplicación. Mediante el uso de este mecanismo se personaliza el acceso y la recuperación de información teniendo en cuenta el usuario. Esta información es utilizada en otras primitivas conceptuales del modelo, como las fórmulas de Filtro o las Condiciones de Navegación.

- **Condiciones de Navegación:** la navegación capturada mediante las relaciones de contexto enlaza unívocamente dos Contextos de Navegación. Sin embargo, dada la naturaleza de las aplicaciones

Web, son necesarios mecanismos que expresen dinámicamente Condiciones de Navegación que sean evaluadas en tiempo de ejecución. Estas condiciones especifican restricciones que se satisfacen para que se produzca la navegación.

- **Navegación con cambio de Usuario:** el Modelo de Navegación no permite definir navegaciones a un contexto o un subsistema del Mapa de Navegación, para el cual el usuario no tiene permisos. En los casos que se desea evitar este comportamiento, es posible definir una Navegación con cambio de Usuario mediante la cual, el usuario conectado adquiere un rol que sí tiene los permisos de acceso correspondientes. Este tipo de navegaciones tienen asociadas un nuevo proceso de identificación, para no violar las restricciones de seguridad.

Una vez definido el Modelo de Navegación que captura la semántica de navegación del sistema, se tienen que asociar características de presentación al sistema. Esta es la función del Modelo de Presentación, que se define en base a un conjunto de primitivas, que especifican como se organiza visualmente la información. El Modelo de Presentación de OOWS se encuentra ligado al Modelo de Navegación, ya que utiliza también los Contextos de Navegación para establecer las propiedades de presentación. Por lo tanto, es una extensión del Modelo de Navegación que enriquece su semántica con un conjunto de patrones de presentación de la información. Estos patrones son utilizados mediante la asociación con los distintos elementos que forman una AIU. Los patrones de presentación de información soportados son:

- **Paginación:** cuando se utiliza este patrón, el conjunto de instancias a recuperar en un Contexto de Navegación es dividido en subconjuntos o "páginas" de un cardinalidad determinada, de tal forma que únicamente se muestra uno de estos conjuntos al usuario. El patrón proporciona, además, los mecanismos necesarios para que el usuario acceda directamente a un conjunto de información.

- **Ordenación:** ordena la población de una AIU según el valor de uno o más atributos de las clases navegacionales que la conforman. El patrón define tanto ordenaciones ascendentes como descendentes.

- **Disposición:** este patrón describe la disposición de la información en la AIU. En OOWS se soportan cinco tipos de disposiciones: tabular (vertical u horizontal), registro, árbol, texto y maestro-detalle.

- **Orden de aparición** (*tabbing*): específica una relación de orden para mostrar tanto los atributos como las operaciones definidas en una AIU.

Mediante estos patrones de presentación y la información definida en el Modelo de Navegación, OOWS captura los requisitos básicos de interacción para la posterior generación sistemática de la IU de una aplicación Web.

2.5 Conclusiones

En este capítulo se han introducido los distintos fundamentos metodológicos en los cuales se ha basado la tesis doctoral. En primer lugar, se ha expuesto la metodología de investigación empleada: *Design Science*. Para cada elemento de la metodología (entorno, base de conocimiento y artefactos) se ha definido, de forma clara y precisa, la contribución de la tesis involucrada. De esta forma, se establece una relación directa entre las contribuciones y el proceso de investigación seguido para obtenerlas.

A continuación se ha proporcionado una visión general tanto del método OOWS como del método OOWS 2.0. Para realizar una comparación detallada se han descrito ambos métodos utilizando un PDD. El uso de esta notación ha resultado útil para establecer un marco formal en el cual realizar la comparación. Definiendo de forma precisa tanto el proceso como las actividades que componen cada método, ha resultado sencillo destacar las similitudes y diferencias entre ambos.

Por último, se han introducido un conjunto de modelos conceptuales que son relevantes en el marco de la tesis. Estos modelos son el Modelo de Presentación de OO-Method y los Modelos de Navegación y Presentación de OOWS. Dada su contrastada utilidad a la hora de modelar la interacción, la

expresividad de sus primitivas conceptuales es considerada en las siguientes secciones.

Capítulo 3

Estado del Arte

En el marco de la tesis doctoral, se propone la definición de un nuevo Modelo de Interacción Abstracto sobre el cual se aplican el resto de contribuciones. Por lo tanto, en la sección 3.1, se ha realizado un breve resumen sobre los trabajos más relevantes pertenecientes a la comunidad Interacción Persona-Ordenador (IPO). A continuación, en la sección 3.2, se han analizado los trabajos directamente relacionados con el desarrollo de aplicaciones Web 2.0. Este apartado se ha estructurado en función de los métodos de Ingeniería Web que han sido extendidos para soportar distintos aspectos relacionados con este tipo de aplicaciones. Para cada método, se han detallado los trabajos relacionados en este ámbito específico. En consecuencia, en la presente tesis no se va a analizar en qué consisten dichos métodos. Un estado del arte detallado sobre diversos métodos de Ingeniería Web y las herramientas que les dan soporte, puede consultarse en el capítulo dos de [Valverde 2007] o con mayor profundidad en [Rossi, Pastor *et al.* 2008]. Las contribuciones de los métodos analizados se centran, fundamentalmente, en el modelado de interfaces RIA, la definición del comportamiento de la interfaz y el uso de patrones en el ámbito de la Web 2.0. Además, se han tenido en consideración otros trabajos que, si bien no se encuentran en el ámbito de los métodos de Ingeniería Web, proporcionan soluciones interesantes en el marco del desarrollo de aplicaciones Web 2.0. En este sentido, la sección 3.3 describe un conjunto de trabajos relacionados con el desarrollo de RIA,

mientras que la sección 3.4, introduce varios trabajos que describen patrones habituales de las aplicaciones Web 2.0. Por último, la sección 3.5 expone las conclusiones sobre el estado del arte realizado.

3.1 Trabajos relacionados con el modelado de la interacción

A continuación, se analizan las propuestas de diversos autores a la hora de modelar la interacción desde las primeras fases de desarrollo del software. La mayoría de estos trabajos abordan la definición de la interfaz de usuario (IU), pero parte de la expresividad propuesta se enmarca en la definición de interacción en la cual se basa la presente tesis. Un buen punto de partida es el trabajo de [Silva & Paton 2003] que proporciona un listado de entornos de modelado de IU. En este trabajo se analiza un total de catorce métodos de modelado de interfaz en base a características como el proceso de desarrollo de la interfaz, los modelos conceptuales que proporcionan o los editores para el diseño e implementación. Algunos ejemplos referenciados en este trabajo son las herramientas Trident [Bodart, Hennebert *et al.* 1994], Genius [Janssen, Weisbecker *et al.* 1993] y MOBI-D [Puerta & Maulsby 1997]. Esta última aproximación es especialmente interesante puesto que utiliza un modelo de tareas y un modelo de diálogo que expresan, de forma implícita, la interacción que representa la IU. Además, enfatiza la comunicación de los requisitos del usuario final con el desarrollador de la interfaz, a fin de garantizar que la interfaz plasme de forma concisa la interacción esperada. Sin embargo, la herramienta para construir los modelos se basa en un editor en forma de árbol y una notación textual poco comprensibles. Por esta razón, la construcción de la interfaz resulta compleja en la práctica.

Una notación ampliamente utilizada en entornos académicos para definir la interacción son los llamados *ConcurTaskTrees* o CTT propuestos por [Paternò 2004]. Un CTT representa una descomposición jerárquica, en forma de árbol, de las tareas que el usuario realiza con la aplicación para alcanzar un objetivo. De esta manera, cada tarea padre del árbol es descompuesta en varias subtareas hijas para describir la interacción al nivel de detalle que desee el analista. La notación define cuatro tipos de tarea: (1) *User Tasks:* tareas que son llevadas a cabo por el usuario sin interaccionar con el SI; (2) *Ap-*

plication Tasks: tareas que son ejecutadas completamente por el sistema; (3) *Interaction Tasks:* tareas iniciadas por el usuario que involucran una interacción con el SI; (4) *Abstract Tasks:* tareas complejas formadas por la composición de las anteriores.

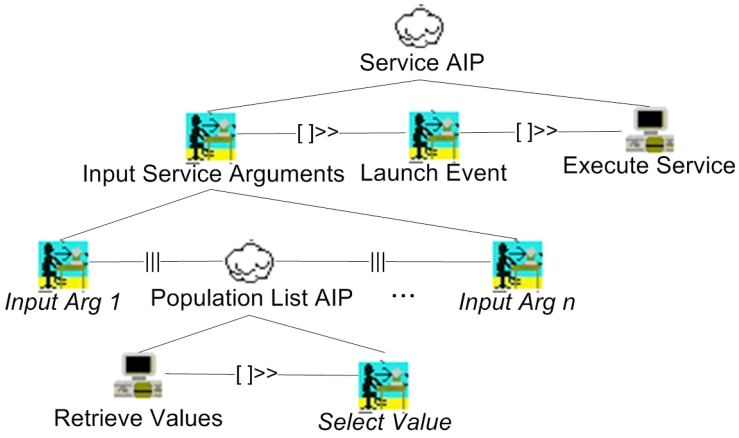


Fig. 3.1: Ejemplo de CTT para la ejecución de un servicio

Adicionalmente, la notación permite establecer relaciones temporales basadas en un conjunto de operadores (*interleaving, synchronization, enabling, enabling with information, deactivation, iteration, finite iteration, optional task* y *recursion*) que son aplicados entre las tareas de un mismo nivel. Así, se describe de forma no ambigua la secuencia de tareas que define la interacción representada por el CTT. Los CTT son una notación muy popular en la comunidad IPO por ser sencillos de comprender y especificar. Además, ofrecen una semántica precisa para evitar ambigüedades en la especificación de la interacción, y han sido validados mediante la especificación de la interacción en aplicaciones industriales. Debido a estas ventajas, diversas aproximaciones han usado los CTT como es el caso de CIAM [Molina, Redondo *et al.* 2008], un método centrado en el desarrollo de IU colaborativas que define un modelo de interacción basado en esta notación.

Otra aproximación relevante presentada en el trabajo de Da Silva es USIXML (USer Interface eXtensible Markup Language) [Vanderdonckt, Limbourg *et al.* 2004], un lenguaje XML para la descripción de interfaces de usuario en múltiples contextos de uso. Las aplicaciones descritas usando este

lenguaje preservan la independencia entre el diseño de la interacción, y las características de la plataforma tecnológica en la cual se implementa la IU. USIXML soporta, a partir de un modelo común, diversos dispositivos (ratón, pantalla, teclado, sistema de reconocimiento de voz), distintas plataformas (teléfono móvil, Pocket PC, Tablet PC, laptop, desktop), y diferentes modalidades de interacción (interacción gráfica, vocal, 3D). La especificación de la interacción en USIXML se hace a dos niveles: partiendo del Modelo de Tareas y Conceptos, se define un modelo abstracto independiente de la plataforma; el modelo abstracto se refina en un modelo concreto para un contexto de uso específico con el fin de obtener la interfaz. Sin embargo, se debe mencionar que USIXML no es un método de producción de software por si solo. Necesita de un motor de transformaciones que interprete el modelo y genere el código de la interfaz. Asimismo, únicamente genera la parte de visual o estática de la interfaz y no cómo ésta se comunica con la funcionalidad del sistema.

Relacionado con USIXML encontramos el entorno IdealXML [Montero, López-Jaquero *et al.* 2005]. Esta herramienta permite la animación de una interfaz abstracta de usuario a través de su modelo de tareas. A partir del modelo de tareas representado mediante un CTT, proporciona una IU que se deriva del mismo y que permite al analista comprobar su funcionamiento. Ya que un modelo de tareas representa la interacción con el sistema, este entorno muestra la interacción en fases tempranas del proceso de desarrollo. Para soportar este proceso de generación, IdealXML proporciona una serie de heurísticas que transforman el modelo de tareas al lenguaje USIXML. Es más, el entorno permite al analista seleccionar una tarea en particular y obtener su especificación de IU correspondiente. Con dicho fin, esta aproximación permite crear prototipos fácilmente modificables y a un nivel de abstracción centrado más en la interacción y la usabilidad de la interfaz, que en su apariencia visual. De esta manera, IdealXML consigue su objetivo principal: la prototipación rápida de interfaces de usuario.

En una línea de razonamiento similar a la de USIXML, [Mori, Paterno *et al.* 2004] proponen el entorno TERESA que se basa en la filosofía "Un solo modelo, varias interfaces". El proceso de modelado de este entorno comienza definiendo un modelo de tareas para una aplicación "nómada", es decir, cuyo funcionamiento varia dependiendo del dispositivo en el cual se utili-

ce. Desde el punto de vista de la interacción, este modelo describe el conjunto de actividades que tienen que ser soportadas para que el usuario pueda alcanzar las metas deseadas a través de múltiples dispositivos. Salvo esta característica propia del entorno, el resto de fases de modelado de la interfaz son idénticas a las de USIXML pero introduciendo elementos de modelado propios. En el nivel abstracto, se introduce el concepto de interactuador que se define como un objeto de interacción abstracto identificado en base a la tarea básica que soporta. A continuación, en el nivel concreto, cada uno de estos interactuadores es reemplazado por un objeto de interacción específico de la plataforma. Estos objetos poseen un conjunto de atributos que definen tanto su apariencia como su comportamiento.

Un trabajo que presenta una notación original para el modelado de la interacción es DiaMODL [Traetteberg, Molina *et al.* 2004]. Esta notación resulta interesante porque describe diálogos de interacción que tienen en cuenta, tanto la gestión de la información como la funcionalidad subyacente. La notación se basa en los conceptos de interactuador (*interactor*) y puerta (*gate*), que representan, respectivamente, la entrada y salida de un objeto abstracto de interfaz. Además, añade los conceptos de variable, computación (*computation*) y conexión (*connection*) para especificar cómo la información es almacenada, transformada y transportada a través de los interactuadores. En consecuencia, DiaMODL define la interacción como un flujo de datos en el cual los interactuadores conectados reciben los datos y, posteriormente, son "devueltos" al resto de la interfaz a través de las puertas. El modelo incluye la definición de variables que almacenen información estática, y computaciones que modifican la información mediante la aplicación de funciones. La notación se encuentra actualmente soportada mediante un editor de prototipos para definir modelos de DiaMODL. Mediante este editor, se asocian clases y objetos UML a dichos modelos, y se define la interfaz de usuario final asociando *widgets* a los distintos elementos del modelo. La contribución principal de este trabajo es que no solo aporta un modelo conceptual propio, sino que también define el comportamiento de la interfaz y como los elementos de modelado se asocian a los *widgets* tecnológicos.

Además de las propuestas ya mencionadas, existen otras basadas en el lenguaje UML como es el caso de WISDOM (Whitewater Interactive System Development with Objetc Models) [Nunes & Cunha 2001], un método

de Ingeniería del Software especializado para la construcción y el mantenimiento de aplicaciones interactivas para PYME. Los distintos modelos conceptuales de WISDOM abarcan el ciclo de desarrollo completo, desde la captura de requisitos hasta la implementación. En el ámbito en el cual nos encontramos, tres de sus modelos están relacionados de forma explícita con el soporte a la interacción: el modelo de interacción de la fase de análisis y los modelos de diálogo y presentación de la fase de diseño.

El modelo de interacción de WISDOM se basa en los conceptos de tarea y espacio de interacción. Una tarea se define como la secuencia de acciones entre el usuario y el sistema para llevar a cabo una meta. Por otra parte, un espacio de interacción define el lugar físico de la interfaz donde se lleva a cabo una tarea específica. Por lo tanto, el modelo de interacción de WISDOM parte con la misma idea de interacción utilizada en la presente tesis. A continuación, para detallar las relaciones temporales entre las tareas, se utiliza el modelo de diálogo. Este modelo es una adaptación de la notación CTT al lenguaje UML y posee la misma semántica que la utilizada en otras aproximaciones como TERESA. Por último, el Modelo de Presentación define las diferentes entidades de presentación, por ejemplo los componentes de un *framework* de diseño de IU, que se encargan de soportar la interacción física con el usuario. Este modelo propone el uso de cinco entidades estereotipadas de UML (*Navigate, Input, Output, Contains* y *Action*) que representan, de forma abstracta, las funciones más habituales en una IU.

En consecuencia, estos tres modelos convierten a WISDOM en una aproximación que soporta de forma precisa el modelado de la interacción. Sin embargo, WISDOM solo aborda el modelado a nivel teórico y no contempla, al mismo nivel de detalle, los aspectos relacionados con la definición de la funcionalidad. Así que, por ejemplo, no da soporte a la generación de la interfaz final o como conectarla con la lógica de negocio.

Otra propuesta interesante basada en UML, es la extensión UMLi propuesta por [Silva & Paton 2003]. UMLi propone una serie de extensiones a nivel de modelado de UML a fin de resolver uno de los inconvenientes tradicionales de este lenguaje: el diseño de interfaces de usuario. En UMLi, la interacción se define a partir de objetos de interacción que son abstractos o concretos, dependiendo si se refieren a una tecnología concreta como, por

ejemplo, la representación de un botón en Java. Al considerar la interacción como un conjunto de objetos del SI, los autores proponen el uso del diagramas de clases UML para describir mediante clases (denominadas *Interaction Class*) tales objetos. No obstante, ya que la representación visual del diagrama de clases no es la más adecuada para modelar la interacción, proponen también una notación alternativa para definir una IU abstracta. Este nuevo diagrama de interfaz de usuario se basa en seis constructores (*free container, editor, action invoker, inputter, displayer* y *editor*), que clasifican la función que realiza un objeto de interacción concreto. UMLi también soporta el modelado de tareas mediante diagramas de caso de uso que son detallados mediante los diagramas de actividades correspondientes. Por lo tanto, la interacción se representa en UMLi, de forma implícita, a través de estos diagramas de actividad en los cuales se definen flujos de comunicación entre los objetos de interacción definidos en el diagrama de IU.

Cabe señalar que UMLi dispone tanto de un editor de modelos como de un generador de código para obtener la IU final. Si bien esta propuesta cubre el ciclo de desarrollo completo, los modelos involucran demasiados detalles razón por la cual el proceso de modelado resulta poco práctico. Este inconveniente provoca que la definición de una especificación UMLi para la IU de una aplicación de tamaño medio sea una tarea compleja. Por esta razón, UMLi no ha gozado de un gran apoyo en ambientes industriales.

Después de analizar las distintas propuestas, se observan ciertas similitudes entre ellas. En concreto, una constante es la separación en dos niveles de abstracción. Con el fin de alinear las distintas propuestas entre si, [Calvary, Coutaz *et al.* 2003] proponen un *framework* unificado desarrollado en el marco del proyecto europeo *Chamaleon*. El objetivo de dicho *framework* es el de establecer un marco de referencia común, a la hora de clasificar los distintos métodos o aproximaciones para el desarrollo de IU, que soportan múltiples contextos o plataformas.

Según dicho *framework*, un contexto de uso se descompone en tres facetas: los usuarios finales del sistema interactivo, la plataforma hardware/software que utilizan los usuarios para realizar la interacción y el entorno físico en el cual se encuentra el usuario. Este *framework* define una IU sensible al contexto como aquella con la capacidad de percibir el contexto y reac-

cionar ante sus cambios. El *framework* propuesto en este trabajo define el ciclo de desarrollo en cuatro niveles de abstracción que se muestran en la Fig. 3.2: tareas y conceptos, interfaz de usuario abstracta, interfaz de usuario concreta e interfaz de usuario final. Los niveles se relacionan entre si mediante una relación de reificación de un nivel más abstracto a uno más concreto. Tomando como base estos cuatro niveles, la mayoría de los trabajos aquí descritos son comparados mediante su alineación con los distintos niveles del *framework*. Por lo tanto, este trabajo resulta esencial para comprender los diversos métodos de modelado de la interfaz, y es una fuente importante de definiciones en este ámbito.

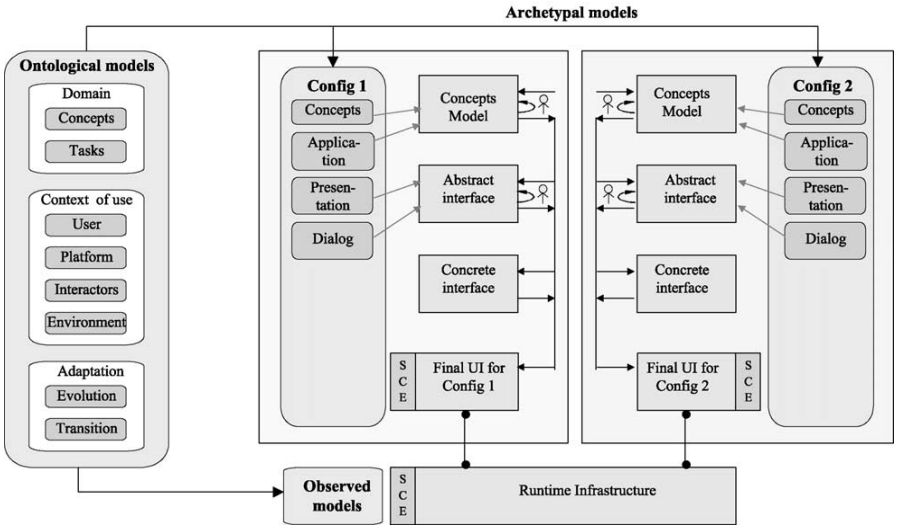


Fig. 3.2: *Framework* de referencia propuesto en el proyecto Chamaleon

Finalmente, conviene analizar como se han tratado los aspectos de interacción en el ámbito específico de la ingeniería Web. Al igual que OOWS, los métodos de Ingeniería Web han introducido modelos conceptuales para capturar de forma abstracta los aspectos estructurales, navegacionales y de presentación de las aplicaciones Web. Por lo tanto, aunque acotados al dominio específico de la Web, también contemplan la generación de una IU a través de la combinación de los distintos modelos conceptuales que soportan la interacción.

Aunque, en líneas generales, los métodos de Ingeniería Web no han representado la interacción con el mismo nivel de detalle que los métodos de la comunidad IPO, las interacciones fundamentales de recuperación de información, navegación y ejecución de servicios se encuentran suficientemente soportadas. El problema fundamental reside en que los métodos de Ingeniería Web se han centrado únicamente en el denominado nivel abstracto. Aunque los modelos de presentación intentan introducir detalles concretos de la IU, estos modelos no han sido especificados con el mismo nivel de formalidad que el resto de modelos. Por ejemplo en UWE y WSDM, los únicos aspectos de la IU que son definidos son aspectos visuales como el color de fuente o la tipografía. Además, estos aspectos no se definen a nivel de modelado sino que utilizan plantillas de estilo en el nivel de implementación. En otras aproximaciones como OOHDM o WebML, los modelos de IU se centran, fundamentalmente, en la distribución de la información en la pagina Web y en la introducción de patrones de presentación de la información. Cabe destacar que ambos inconvenientes también se encuentran presentes en el método OOWS actual.

En resumen, actualmente, los métodos de Ingeniería Web no tienen en cuenta el nivel concreto de la interacción. Este era un problema menor en las aplicaciones Web que se desarrollaban hasta ahora. No obstante, en el desarrollo de aplicaciones Web 2.0, resulta un factor determinante el uso de las características más avanzadas que proporciona la tecnología, con el fin de definir interacciones más atractivas al usuario. Sin disponer de un modelo concreto enlazado con los modelos abstractos actuales, toda la problemática tecnológica queda relegada, casi por completo, al nivel de implementación.

A la hora de definir el Modelo de Interacción Abstracto, se siguen las ideas que se definen en el *framework* de referencia *Chamaleon*, que utilizan varios métodos de la comunidad IPO. Estas ideas son: el énfasis de la interacción como realización de tareas por parte del usuario, la separación de la interacción en dos niveles de abstracción y la posibilidad de definir elementos de interacción independientes de la tecnología. De esta manera, la tesis introduce mecanismos para modelar como el usuario interactúa con la IU y se interrelaciona con el sistema de información. Por dicha razón, conviene recalcar el concepto de *Interacción* ya que, en este nivel abstracto, se obvia la parte visual de la interfaz centrándonos en el comportamiento de la misma.

Asimismo, el modelo de interacción propuesto se integra con el resto de modelos del método OOWS 2.0, de tal forma que es factible definir la interacción de un sistema de información funcional.

3.2 Trabajos relacionados en el ámbito de los métodos de Ingeniería Web

En el ámbito de la Ingeniería Web, se han propuesto varios métodos para abordar el desarrollo de aplicaciones Web. Con el auge de las aplicaciones Web 2.0, varios de ellos han sido extendidos para soportar la expresividad necesaria. Fundamentalmente se ha hecho un especial énfasis en la definición de modelos para el soporte de *Rich Internet Applications* (RIA), al ser ésta la contribución más visible a nivel tecnológico de la Web 2.0. A continuación, se detallan los trabajos realizados en el ámbito de la Ingeniería Web y como éstos mejoran el desarrollo de las aplicaciones Web 2.0.

3.2.1 WEBML

El método WebML [Ceri, Fraternali *et al.* 2003] fue el primero en introducir, a nivel conceptual, la expresividad necesaria para el modelado de RIA. Al igual que otros métodos de Ingeniería Web, WebML se basa en una serie de modelos conceptuales para especificar las distintas facetas de una aplicación Web. Sin embargo, debido a las carencias detectadas para modelar RIA, los autores proponen una extensión al método [Bozzon, Comai *et al.* 2006]. A continuación, se resumen para cada uno de los modelos de WebML las extensiones realizadas:

- **Modelo de datos:** para el modelado de datos el método utiliza notaciones clásicas como los diagramas Entidad-Relación o los diagramas de clases UML. Sin embargo, en el dominio de las RIA, se tienen que considerar dos dimensiones adicionales para caracterizar correctamente los datos: (1) donde reside físicamente la información, es decir, en la parte cliente (el navegador Web) o en la parte servidor (generalmente una base de datos) y (2) el nivel de persistencia, el cual puede ser temporal o volátil. Para capturar dichas dimensiones, los

autores extienden el modelo de datos mediante estereotipos que indican, para cada entidad o relación, un valor para estas dos dimensiones. Asimismo se imponen una serie de restricciones ya que, por ejemplo, una entidad que reside en la parte cliente no puede relacionarse con una entidad que reside en la parte servidor.

- **Modelo de navegación:** Utilizando el modelo de datos, WebML especifica varios *Site Views* que se definen como un grafo de páginas Web organizadas en una o varias áreas, pero con una misma finalidad. En el ámbito de las RIA, el concepto de página es ambiguo puesto que, habitualmente, la aplicación reside en un contenedor principal sobre el cual se van cargando los distintos contenidos acorde con la interacción del usuario. Nuevamente, los autores proponen una diferenciación entre *Server Pages,* para especificar las páginas Web renderizadas por la parte servidora, y *Client Pages*, para representar páginas en donde la presentación y el control de eventos son llevados a cabo en la parte cliente. También se permite la anidación de ambos tipos de páginas en una página contenedora. En dichos anidamientos es posible establecer si todas las páginas son procesadas simultáneamente o, únicamente, cuando son demandadas.

- **Modelo de contenidos:** En WebML cada página se compone de una o varias unidades que desempeñan una función determinada como la recuperación de información, la ejecución de una operación o la activación de una navegación. Ya que las unidades se encuentran definidas dentro de una página, es posible derivar si son procesadas en la parte cliente o servidora. Para las unidades cliente es posible además: (1) publicar contenido proveniente de entidades cliente y, (2) establecer condiciones de filtrado o de ordenación sobre las instancias presentes en la parte cliente, sin tener que invocar a la parte servidora.

- **Modelo de operaciones:** En WebML, las operaciones modelan la ejecución de la lógica de negocio y una serie de modificaciones predefinidas sobre el contenido (creación, actualización, borrado y creación de relaciones). Nuevamente, las operaciones son caracterizadas en función si son ejecutadas en la parte servidora o cliente, pero aña-

diendo la definición de secuencias de operaciones encadenadas que mezclen ambos niveles. Un aspecto importante, también contemplado, es la transferencia de información entre la parte cliente y servidora. Con dicho fin se introducen dos operaciones predefinidas: *XMLIn* y *XMLOut*. La primera toma una representación XML de un conjunto de datos transformándola en un conjunto de entidades/relaciones, mientras que la segunda realiza la función inversa.

La extensión a nivel conceptual de WebML se resume en determinar qué primitivas conceptuales son procesadas en la parte cliente, y cuales en la parte servidora. Es decir, se define de forma explícita, a nivel conceptual, la distribución arquitectónica que es habitual en las RIA. El principal inconveniente de esta aproximación es la introducción del modelado de conceptos arquitectónicos, en modelos que tratan otro tipo de problemática. Por lo tanto, se produce una mezcla de diferentes facetas de modelado que siempre es recomendable evitar. También, se propone una aproximación para definir la comunicación entre la parte cliente y servidora mediante representaciones en XML. Sin embargo, esta aproximación no se hace extensible al caso que la parte servidora sea un servicio REST externo, sobre el cual no se dispone de un modelo. En las aplicaciones Web 2.0, este tipo de comunicación basada en servicios REST es la más frecuente.

Otro trabajo relevante realizado en el marco de WebML, propone como los eventos que se producen en una RIA son incluidos en sus modelos conceptuales [Carughi, Comai *et al.* 2007]. Este trabajo propone un Modelo de Eventos para: (1) modelar los distintos tipos de eventos que se producen en una Interfaz RIA y, (2) relacionar las ocurrencias de un evento con las entidades del modelo de datos. El Modelo de Eventos (representado mediante un diagrama E-R) se compone de *Event Entities*, cuyas instancias representan la ocurrencia de un evento concreto de la aplicación. Dichas *Event Entities* incluyen parámetros con información útil en el dominio de la aplicación a desarrollar. Para soportar la notificación de eventos, el Modelo de Navegación de WebML es extendido con dos operaciones adicionales, *send event* y *receive event*, que representan, respectivamente, la notificación del envío o recepción de un evento. Ambas operaciones se relacionan con una *Event Entity* de tal forma que, las operaciones *send* y *receive* definidas sobre la misma *Event Entity,* se comunican entre si. Asimismo también se asocian a un *reci-*

pient que representa una entidad del modelo de datos que recibe las notificaciones. Las reacciones a eventos, que se reciben en la parte servidora, son modeladas mediante una sucesión de operaciones que es desencadenada por una operación *receive-event*. Cualquier operación del servidor puede ser iniciada a consecuencia de dicha notificación.

Gracias a esta aproximación, la definición de eventos a nivel conceptual se encuentra perfectamente ligada a los modelos previos. La principal ventaja es la definición de comportamientos complejos mediante la combinación de las nuevas primitivas, con las ya disponibles. No obstante, el principal inconveniente detectado es que, en el ámbito de las RIA, los eventos relevantes a procesar son aquéllos que se producen a nivel de la IU. Por lo tanto, la definición de estos eventos está muy ligada a la tecnología con la cual ha sido implementada dicha interfaz. La aproximación introducida en WebML caracteriza los eventos como los sucesos relevantes en el dominio de la aplicación, como por ejemplo la asignación de una tarea al usuario, no teniendo en cuenta por lo tanto estos eventos tecnológicos. Por otro lado, la extensión está muy ligada al lenguaje WebML con lo cual es difícil determinar si es aplicable en otros métodos.

Otro aspecto que también ha sido tratado en el ámbito de WebML, es como expresar patrones utilizados en aplicaciones Web sociales [Fraternali, Tisi *et al.* 2009]. En dicho trabajo se analizan diez de las aplicaciones Web sociales más populares, a partir de las cuales se especifica un conjunto de catorce patrones. Para cada patrón se detalla, adicionalmente, las aplicaciones donde es utilizado analizando la relevancia real de cada uno. Asimismo, estos patrones han sido clasificados en base al concepto social que enfatizan, como puede ser la valoración de la reputación de un usuario o la organización de los contenidos de una Web. Por ejemplo, para obtener la relevancia de un determinado contenido se propone el uso del patrón *Rating* (ver Fig. 3.3) que establece un valor numérico a partir de las opiniones de los usuarios.

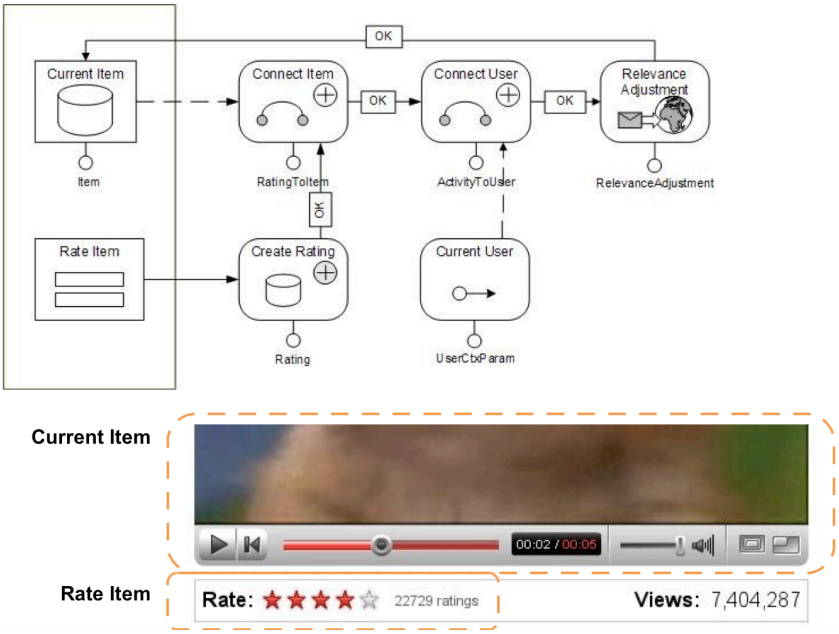


Fig. 3.3: Patrón *Rating* en WebML según [Fraternali, Tisi *et al.* 2009]

Todos los patrones son especificados utilizando modelos WebML, con lo cual la principal contribución de este trabajo es una descripción conceptual de los mismos. Un patrón en WebML no es más que un modelo compuesto de una serie de unidades predefinidas para cumplir una tarea concreta, tal y como muestra la Fig. 3.3 (arriba). Para instanciar un patrón, los elementos que se relacionan con el modelo de datos (entidades, atributos y relaciones) son considerados como roles, que son aplicados sobre elementos de la aplicación final. Respecto a las navegaciones representadas en el patrón, se tienen que seguir una serie de restricciones para asegurar el cumplimiento de la semántica del patrón.

El hecho de especificar los patrones mediante WebML no es un inconveniente, porque la notación se utiliza únicamente con fines documentales. Por otra parte, los modelos son lo suficientemente expresivos como para modelar tanto la parte estática (datos a almacenar) como la parte dinámica (interacción con el usuario) del patrón. No obstante, no se introduce una visión más detallada del patrón, por ejemplo, mediante una plantilla.

Si bien a lo largo del trabajo se justifica que WebML es lo suficientemente expresivo como para soportar dichos patrones, tampoco se presenta un mecanismo para evitar la creación de los modelos correspondientes cada vez que desea utilizarse un patrón. Otro inconveniente es que se aporta una solución específica para el método, y no se aborda una solución más genérica que pueda ser aplicada en otros métodos. A pesar de ello, tal y como los autores indican, es cierto que éste es un escollo menor dada la similitud existente entre los modelos conceptuales utilizados por la mayoría de los métodos de Ingeniería Web.

3.2.2 RUX-Method

El método RUX [Linaje, Preciado *et al.* 2007a] proporciona una serie de modelos para la definición de IU en diversas tecnologías RIA. Su principal ventaja es que se centra, exclusivamente, en el modelado de la Interfaz RIA proporcionando una amplia expresividad en este dominio. Siguiendo los principios de la comunidad IPO, RUX divide el proceso de desarrollo de la interfaz en tres niveles tal y como se muestra en la Fig. 3.4:

- **Nivel Abstracto**: en este nivel, RUX define un modelo de interfaz abstracta que juega el rol de modelo común para las distintas tecnologías RIA, y que no tiene en cuenta aspectos específicos del dispositivo final. Este modelo se define en base a tres entidades principales: (1) *Connectors,* establece la relación entre el modelo de datos y el elemento de la interfaz que muestra dichos datos; (2) *Media*, representa un elemento de información atómica (texto, imagen, video, etc.); (3) *View*, representa un grupo de información que es mostrado simultáneamente al usuario.

- **Nivel Concreto:** RUX ofrece tres modelos para representar tanto la presentación concreta de la IU como su comportamiento. (1) *Spatial Presentation Model,* define el posicionamiento, tamaño y apariencia de los distintos componentes de interfaz. (2) *Temporal Presentation Model*, representa el comportamiento de la interfaz que se produce sin que medie la intervención del usuario. Dicho comportamiento se define mediante *Temporal Relationships*, que establecen el orden

temporal entre distintos componentes de interfaz al invocar la lógica de negocio como consecuencia de eventos temporales. (3) *Interaction Presentation Model,* captura el comportamiento que se produce en la interfaz como consecuencia de los eventos generados por el usuario. Como lenguaje para especificación de dichos eventos se utiliza el estándar *XML Events* de la W3C. Cada evento es a su vez asociado a un *handler* que incluye una regla ECA para especificar las consecuencias del evento.

- **Nivel de Interfaz Final:** este nivel establece la transformación entre la interfaz definida en el nivel concreto y la interfaz final de la plataforma tecnológica seleccionada.

RUX realmente es una extensión metodológica para la generación de IU avanzadas más que un método de Ingeniería Web en si mismo. La principal razón es que este método no proporciona modelos para la definición de las capas de persistencia y de lógica de negocio. Sin embargo en el trabajo desarrollado por [Linaje, Preciado *et al.* 2007b] se propone como acoplar el método RUX con el método WEBML. Por otro lado, los mismos autores [Preciado, Linaje *et al.* 2008] abordan la misma integración pero con el método UWE. Este proceso de integración se lleva a cabo definiendo una serie de reglas de conexión entre el Modelo de Interfaz Abstracta de RUX y los modelos del método a extender. En concreto, de dichos modelos se extrae la información sobre las entidades de datos y sus relaciones, las agrupaciones en páginas de los distintos elementos y la navegación. De esta manera, combinando RUX con un método de Ingeniería Web, es factible definir completamente una aplicación Web 2.0.

La extensión de un método de Ingeniería Web mediante el método RUX, implica establecer conexiones entre los modelos del método y su nivel abstracto de interfaz. Sin embargo, determinados métodos de Ingeniería Web disponen de primitivas de modelado que son equivalentes a las del nivel abstracto de RUX. Como resolver este tipo de inconsistencias no es discutido por los autores. Por otra parte, la integración de RUX se realiza a nivel conceptual, pero no se propone ningún mecanismo para llevar a cabo dicha integración a nivel de implementación. Este punto ha sido abordado por los autores mediante la integración explícita con la herramienta *WebRatio*, si bien

un mecanismo de integración con otras herramientas no ha sido contemplado.

Respecto al nivel concreto de interfaz, el número de *widgets* o componentes gráficos está limitado a un subconjunto seleccionado a partir de varias tecnologías RIA. Si bien este conjunto es lo suficiente representativo como para soportar un amplio conjunto de interfaces RIA, los componentes más avanzados de interfaz, que no se encuentran soportados son aquéllos que realmente aportan las ventajas significativas. Los autores indican que dicho catálogo puede ser extendido conforme a las necesidades del analista, pero no se describe como se realizan dichas extensiones

Por último, el método introduce el *look & feel* en las fases tempranas de desarrollo, a diferencia de otras aproximaciones en el ámbito de la Ingeniería Web, en donde se realiza de forma independiente. Habitualmente, dicha tarea es llevada a cabo por diseñadores ajenos al desarrollo de la aplicación Web, ya que no es recomendable introducir estos aspectos en el proceso de desarrollo.

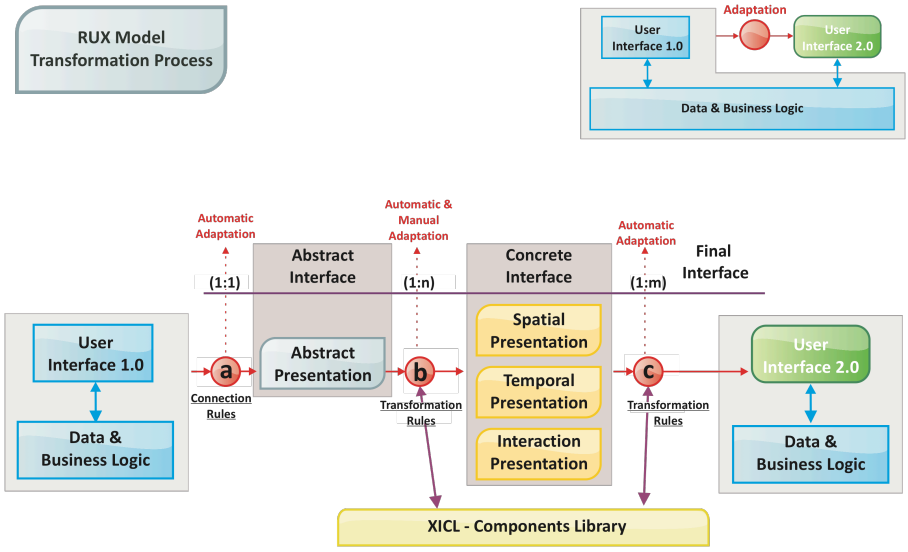


Fig. 3.4: Proceso de desarrollo de RUX-Method

3.2.3 OOH4RIA

Para soportar el desarrollo dirigido por modelos de RIA en el método OOH, se han propuesto dos modelos: un modelo de presentación [Melia, Gomez *et al.* 2008] y un modelo de orquestación [Pérez, Díaz *et al.* 2008]. La inclusión de ambos modelos, junto con las transformaciones necesarias para la generación de código, define un nuevo método de desarrollo que recibe el nombre de OOH4RIA.

El modelo de presentación se basa en una representación estructural de los distintos *widgets* que constituyen una IU. Por lo tanto, la presentación se define mediante un modelo específico de dominio, que persigue la mayor similitud posible con los conceptos que definen las interfaces RIA. Dicho modelo ha sido formalizado mediante un metamodelo MOF, que representa tanto la topología de los *widgets* como sus propiedades y restricciones.

Una IU se representa, utilizando este modelo, como una composición de distintos *screenshots* que contienen un conjunto de *widgets* visibles. Cada *screenshot* se comporta como un contenedor para definir la distribución espacial de los *widgets*. La primitiva conceptual central del modelo es por lo tanto la clase *widget,* que es especializada en los distintos *widgets* concretos del *framework* GWT (Google Widget Toolkit). Dicho *framework* tecnológico se utiliza como base para definir el metamodelo de presentación para OOH4RIA que se muestra en la Fig. 3.5. En caso de ser necesario utilizar un *widget* que no se encuentra definido por dicho *framework,* se extiende el metamodelo mediante especializaciones de la clase *CustomWidget*.

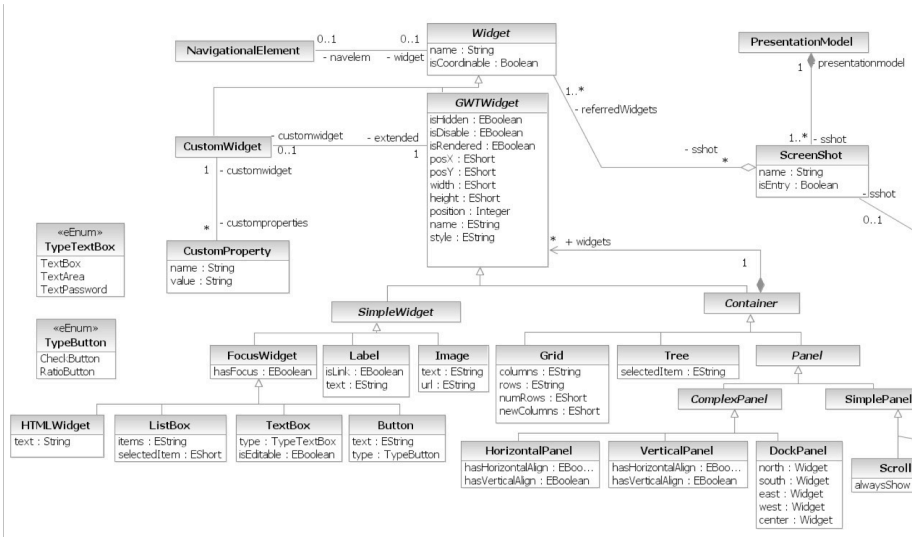


Fig. 3.5: Metamodelo de presentación de OOH4RIA

Por otro lado, el modelo de orquestación es un perfil UML, basado en el diagrama de estados, que captura tanto la interacción con los *widgets* del modelo de presentación como la navegación entre los distintos *screenshots*. Este modelo define *orchestral widgets* que reciben los eventos generados por el usuario y que desencadenan una reacción. Tanto los *orchestral widgets* como los *screenshots* son modelados como estados estereotipados, tal y como muestra la Fig. 3.6.

Los *widgets* reaccionan a consecuencia de los eventos producidos por otros *widgets*. Para capturar estas dependencias de interacción se introduce el concepto de *orchestral transition*. Estas transiciones entre estados se definen mediante una regla ECA (Evento-condición-acción), asociada a la ocurrencia de un evento producido sobre un *widget*. La regla se especifica con el esteriotipo *SignalBroadcast* para comunicar la transición al resto de *widgets* o como *SignalHandler,* si el objetivo es esperar una determinada transición para que se produzca la regla.

Sin embargo, estas transiciones no se producen siempre de manera automática ya que, habitualmente, es necesaria una interacción adicional por parte del usuario. Para soportar estas situaciones se introduce el concepto de

orchestral states, estados en los que es necesario la interacción del usuario para confirmar la transición, para recibir una alerta o para introducir un valor.

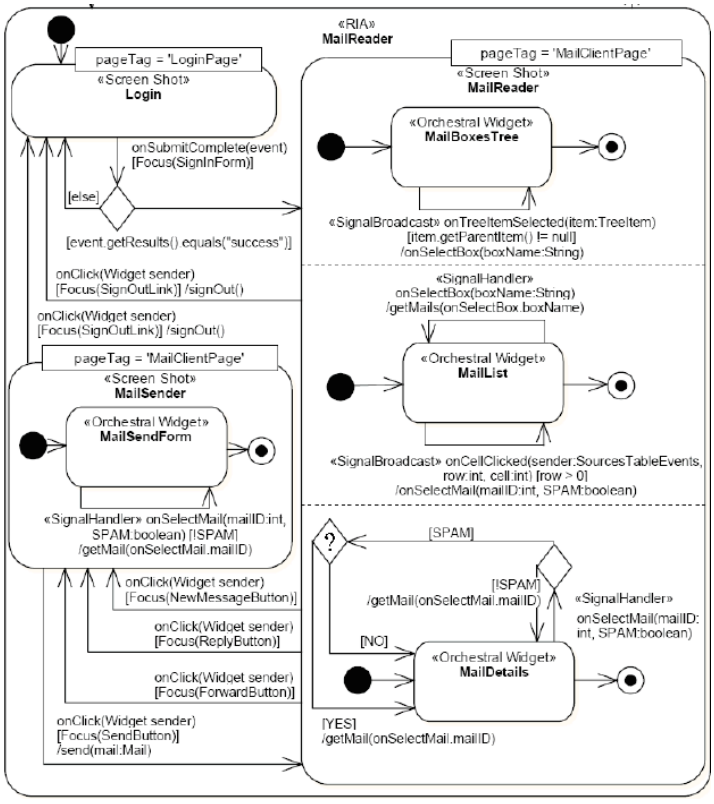


Fig. 3.6: Ejemplo del Modelo de Orquestación de OOH4RIA

Mediante ambos modelos, el método OOH4RIA cumple con la expresividad necesaria para la generación de una Interfaz RIA. La parte estática es definida utilizando un mecanismo, ampliamente aceptado por los diseñadores gráficos, como es la composición mediante *widgets* mientras que la parte dinámica o de comportamiento, se basa en una notación también ampliamente aceptada como son los diagramas de estados UML. El principal inconveniente detectado en el modelo de presentación es su estrecha relación con el *framework* tecnológico GWT. Aunque es cierto que los distintos *frameworks* RIA comparten la gran mayoría de *widgets* y conceptos arquitectónicos, los autores no ponen de manifiesto si el modelo es fácilmente traslada-

ble a otra tecnología utilizando únicamente la entidad *CustomWidget*. Respecto al modelo de orquestación, la amplia expresividad proporcionada, conlleva el inconveniente de una notación gráfica muy compleja, tal y como muestra la Fig. 3.6. Dicho inconveniente hace patente la necesidad de utilizar otro tipo de notación más precisa y sencilla para esta tarea.

3.2.4 OOHDM

En el marco del método OOHDM [Schwabe 1998], han sido propuestas por [Urbieta, Rossi *et al.* 2007] una serie de extensiones para el soporte de interfaces RIA. La principal contribución es la introducción de las denominadas *Abstract Data Views* (ADV) [Cowan & Lucena 1995] que expresan, con un elevado nivel de abstracción, la estructura y el comportamiento de la interfaz.

Una ADV especifica la relación entre los distintos objetos de la interfaz y los objetos de la aplicación. De esta forma, una ADV "observa" un objeto de la aplicación en donde reside cierta parte de los datos y de la funcionalidad del SI. En otras palabras, define cómo dicho objeto de la aplicación es percibido por el usuario. Estas relaciones con los objetos de la aplicación se representan mediante diagramas de configuración, similares a los diagramas de clases UML, que especifican los mensajes que intercambian la interfaz y la lógica de negocio.

Para especificar el comportamiento de la interfaz se introduce el concepto de *ADV-Chart*: un diagrama de transición de estados que define el efecto sobre las ADV como consecuencia de la interacción con el usuario. Los *ADV-Charts* también son utilizados para definir que eventos inician una navegación o la ejecución de una operación. Las distintas transiciones que forman un *ADV-Chart* se componen de una pre-condición, el evento que dispara la transición y una post-condición (el disparo de un nuevo evento, la llamada a un método, etc.). La Fig. 3.7 muestra una ADV, junto con su *chart* asociado, que define una interfaz para la visualización de videos *on-line*.

En el marco de las RIA, las ADV ofrecen una amplia expresividad para especificar distintos tipos de comportamiento. A pesar de ello, la principal limitación de esta aproximación es que proporciona una visión muy abstracta

de la interfaz. Como consecuencia, la correspondencia entre la ADV y la Interfaz RIA a generar no es del todo precisa. Este hecho provoca que las ADV no sean una buena solución para un método que tenga como fin la generación del código final, porque es necesario eliminar la ambigüedad en las reglas de transformación. Los autores tampoco comentan como esta aproximación puede ser aplicada en otros métodos que no sean OOHDM.

Tomando como base el concepto de ADV, el trabajo presentado por [Rossi, Urbieta *et al.* 2008] propone el uso del concepto de refactorización para evolucionar las aplicaciones Web tradicionales hacia RIA. Con dicho fin, los autores introducen el concepto de Refactorizaciones RIA: una serie de patrones de modelado con el objetivo de mejorar la interacción con el usuario en el marco de las RIA. Una Refactorización RIA es aplicada de tal forma que la interacción modelada es transformada o son incluidos nuevos mecanismos de interacción. Para definir una Refactorización RIA, se utiliza la plantilla de una ADV la cual recibe como parámetros los elementos de la interfaz (un campo de texto, un formulario, etc.) que son objeto de la refactorización. Sobre dichos elementos es aplicada tanto la estructura como el comportamiento definido en el ADV. Por ejemplo, la refactoriación "Añadir información sobre el enlace de destino" modifica una navegación de la aplicación para que se muestre antes de ejecutarla, detalles acerca de la página destino. Cada Refactorización RIA se define a partir del problema que motiva su uso y de su mecánica la cual define, de modo preciso, cómo es llevada a cabo la transformación.

Esta aproximación propone una forma de aplicar "Patrones RIA" en un método de Ingeniería Web puesto que las ADV, que definen las refactorizaciones, son modelos conceptuales que abstraen un problema recurrente. Además, estas refactorizaciones son configurables ya que aceptan distintos elementos de interfaz como parámetros. El principal inconveniente, tal y como también destacan los propios autores, es el uso del concepto ADV, que no se encuentra ampliamente extendido en el ámbito de la Ingeniería Web.

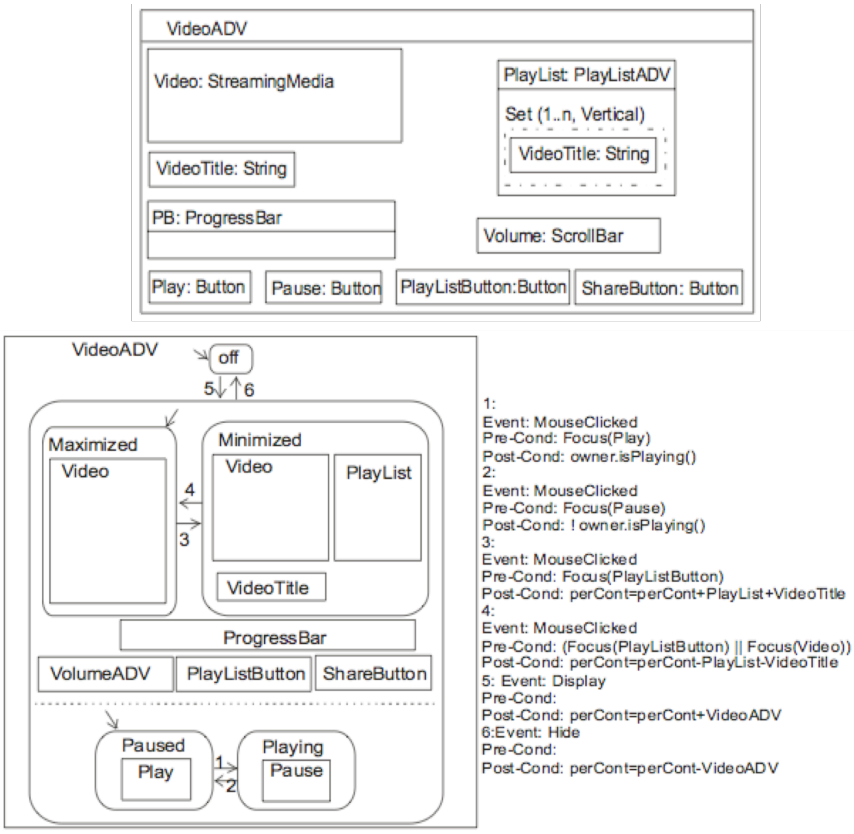


Fig. 3.7: ADV definida en [Rossi, Urbieta *et al.* 2008]

En el ámbito de OOHDM, también se ha discutido el uso de patrones en el sentido estricto del término. En el trabajo de [Rossi 2000] se presentan una serie de patrones que solucionan problemas recurrentes, tanto desde el punto de vista de la navegación como de la interfaz de usuario. En concreto, se analizan los patrones: *Information On-Demand*, *Information-Interaction Decoupling*, *Information-Interaction Coupling*, *Behavioural Grouping*, *Behaviour Anticipation* y *Process Feedback*. Para cada patrón se detalla de forma textual el problema, la motivación, la solución y los usos conocidos, mediante una serie de ejemplos. Los patrones analizados tratan problemas relacionados con la interfaz y el comportamiento de la misma. No obstante, los autores solo proporcionan unas guías generales de cómo aplicar dichos patrones esbozando, brevemente, cómo podrían ser modelados e introducidos en el

método de Ingeniería Web OOHDM. Por otro lado, dichos patrones están enfocados a buenas prácticas de la ahora denominada "Web 1.0". A pesar de ello, el trabajo pone de manifiesto las ventajas de introducir el uso de patrones en el ámbito de la Ingeniería Web. Por esta razón, gran parte de las ideas planteadas pueden ser aplicadas en el dominio de la Web 2.0.

3.2.5 UWE

Siguiendo una línea de razonamiento similar a los trabajos ya comentados, [Koch, Pigerl *et al.* 2009] proponen una aproximación basada en patrones para el desarrollo de RIA dirigidas por modelos. Dada la estrecha relación existente entre las RIA y las aplicaciones Web 2.0, los patrones que se proponen en dicho trabajo -que son concretamente los de Autocompletado y Actualización dinámica- son habituales en las aplicaciones Web 2.0. Para modelar los patrones, este trabajo propone: (1) un lenguaje de eventos, para representar la interacción entre el usuario y el sistema, y (2) un diagrama de estados UML, para representar los efectos de los eventos sobre la aplicación Web. Asimismo, los autores proponen que dichos diagramas de estado sean integrados en un método de Ingeniería Web mediante la definición de *extensión points*, relaciones de las primitivas conceptuales que representan el patrón a nivel de metamodelo y reglas de transformación de modelo a código. De esta forma, el comportamiento abstraído se genera automáticamente. Los autores aplican dicha estrategia en el marco del método UWE en el cual añaden una serie de atributos, clases y relaciones a nivel de metamodelado. Estos nuevos elementos de metamodelado son representados, posteriormente cuando se crea un modelo, mediante etiquetas sobre las primitivas conceptuales correspondientes.

El trabajo presenta dos limitaciones: en primer lugar, se centra únicamente en abstraer patrones de comportamiento de la interfaz habituales en las aplicaciones Web 2.0. Si bien estos patrones proporcionan unas ventajas evidentes al analista, se tienen en cuenta únicamente buenas prácticas desde la perspectiva de la Interfaz RIA. Desde nuestro punto de vista, nos parece relevante incluir patrones que también enfaticen y simplifiquen la colaboración del usuario en el marco de la aplicación. Por otro lado, la estrategia de integración a nivel de metamodelado no está descrita con el suficiente rigor, como para ser trasladada a otros métodos de Ingeniería Web. Aunque en el

marco de UWE resulta viable, como bien ponen de manifiesto los autores, no se discute si realmente la misma extensión a nivel de modelado es factible en otros métodos.

3.2.6 Análisis comparativo

Con el objetivo de delimitar cuales son las contribuciones específicas de la presente tesis, a continuación se realiza una análisis comparativo sobre las aportaciones y carencias de cada uno de los métodos analizados. En esta línea, [Preciado, Linaje *et al.* 2005] han realizado un análisis similar, pero contemplando solo los aspectos relacionados con el desarrollo de RIA. Además, se tienen que considerar los progresos realizados en los últimos años en esta línea de investigación.

Específicamente, el análisis se ha centrado en cuatro características representativas que forman parte de las contribuciones de la presente tesis. Cabe destacar que algunos de estos métodos ofrecen contribuciones relevantes no consideradas en el presente análisis como, por ejemplo, la modelización de la funcionalidad y la persistencia en la distintas capas de la aplicación [Bozzon, Comai *et al.* 2006] o el soporte a nivel de herramienta. Al quedar estos aspectos fuera del alcance de la presente tesis, no han sido tenidos en cuenta para acotar, de forma precisa, la mejora que proporciona el método OOWS 2.0. Las características analizadas son enumeradas y justificadas a continuación.

1) Modelado abstracto de la interacción

Una de las principales ventajas de las aplicaciones Web 2.0 es la sencillez con la cual se define la interacción con el usuario. Entendemos la interacción en el contexto de esta tesis, como la secuencia de acciones que realiza el usuario a través de la interfaz para obtener o proporcionar información al SI. Los métodos de Ingeniería Web han representado este concepto de interacción utilizando fundamentalmente modelos navegacionales. La navegación era, hasta ahora, el mecanismo de interacción principal que se producía en el marco de las aplicaciones Web. Sin embargo, en la Web 2.0, otras interacciones adquieren un rol igual de relevante. Básicamente, un método de Ingeniería Web tiene que soportar un modelado de la interacción más complejo

que el actualmente considerado. Este modelado de la interacción tiene que realizarse, en una primera instancia, de modo abstracto para que el analista se centre en la definición del proceso interactivo con el usuario, sin tener que preocuparse por aspectos de carácter tecnológico.

Respecto al modelado de la interacción, el soporte que dan los métodos de Ingeniería Web es elevado ya que, en este sentido, un gran número de las primitivas conceptuales propuestas previamente pueden ser reutilizadas. Este es el caso de los métodos WebML, OOH y OOHDM. No obstante el método UWE, debido a que está basado en UML, hereda los problemas de este lenguaje a la hora de modelar de forma adecuada la interacción. Los diagramas de interacción de UML se centran en la comunicación entre los distintos objetos de las clases, mientras que la interacción planteada en el contexto de esta tesis se sitúa a un nivel de abstracción mucho mayor, entre el usuario y el SI. Por otra parte, RUX, al ser un método más reciente, se ha centrado principalmente en los aspectos concretos de la IU. La interacción con el SI no ha sido tratada implícitamente, sino mediante la conexión con otros métodos.

2) Diseño de interfaces de usuario RIA

Desde el punto de vista tecnológico, el desarrollo Web ha evolucionado considerablemente. A raíz de esta evolución, han surgido una serie de tecnologías que permiten desarrollar IU más ricas tanto desde el punto de vista visual como interactivo. Como se ha discutido anteriormente, esta mejora tecnológica a dado lugar a las llamadas *Rich Internet Applications* o RIA. La mayoría de las aplicaciones Web 2.0 son, actualmente, desarrolladas utilizando dichas tecnologías. Por lo tanto, un gran número de las aplicaciones Web 2.0 actuales son ejemplos claros de RIA. Los métodos de Ingeniería Web no habían puesto hasta el momento mucho énfasis en la expresividad de los modelos de presentación relegando, incluso a menudo, los aspectos de presentación al nivel de implementación. Con el auge de la tecnologías RIA, es un requisito fundamental que el diseño de la IU esté soportado por el propio método considerando, además, los aspectos concretos de la tecnología seleccionada.

El soporte al desarrollo de IU basadas en tecnologías RIA ha sido un aspecto ampliamente abordado por los distintos métodos. El método RUX

ha realizado el mayor énfasis en este aspecto, puesto que proporciona una serie de modelos específicos para abordar gran parte de esta problemática. Asimismo, mediante la implementación de la herramienta *RUX-Tool* se ha tratado como generar la interfaz para distintas tecnologías RIA. Aunque el método WebML fue el primero en tratar la problemática de las RIA, fundamentalmente, se ha centrado en distinguir qué entidades de los modelos con implementadas en la capa cliente y cuáles en la capa servidor. A nivel de diseño de interfaz se introducen, únicamente, unas pequeñas mejoras en su modelo de presentación. En el método UWE ocurre una situación similar ya que relega, al nivel de implementación, muchos aspectos del diseño de la Interfaz RIA.

Por otra parte, OOH4RIA define un nuevo metamodelo de presentación basado en el *framework* RIA *Google Web Toolkit* (GWT). De esta manera, el método es capaz de capturar la expresividad de este nuevo tipo de interfaces complejas. Además, el proceso de generación de código se simplifica debido a que los conceptos utilizados en el nivel de diseño y en el de implementación son similares. Sin embargo, no se aborda en detalle como dicho metamodelo puede adaptarse a otras tecnologías RIA. Es cierto que GWT proporciona una serie de elementos de interfaz genéricos que son comunes entre distintas tecnologías. A pesar de ello, es precisamente en las diferencias donde residen las principales ventajas de cada tecnología RIA. Por último, OOHDM da un soporte parcial a este aspecto introduciendo un nivel concreto en el modelo de interfaz. No obstante, no se detalla que elementos conforman dicho nivel concreto ni como se lleva a cabo, si de manera manual o sistemática, la correspondencia de las ADV con dicho nivel.

3) Comportamiento de la interfaz dirigido por eventos

En las aplicaciones Web 2.0, las IU son muy reactivas en función de los eventos que produce el usuario (hacer clic sobre un elemento de la interfaz, desplegar un menú, etc.). Este comportamiento dirigido por eventos es muy habitual en las IU de las aplicaciones de escritorio. Sin embargo, el lenguaje HTML 4, que es la base principal para definir interfaces Web, carece del soporte adecuado para implementar este tipo de comportamientos de interfaz complejos. Con la llegada de las tecnologías RIA, esta carencia ha sido solventada siendo posible incluir una capa de comportamiento lógico en el nivel

de interfaz. Hasta el momento, los métodos de Ingeniería Web no habían tenido en cuenta la definición de este comportamiento de la interfaz, a diferencia de los métodos IPO para el desarrollo de IU.

El comportamiento de la interfaz es otro aspecto tratado en detalle en los métodos RUX, OOHDM y OOH4RIA. Los dos primeros siguen una aproximación bastante similar basada en el concepto de regla ECA, es decir, definen a partir de los eventos de interfaz una serie de acciones a llevar a cabo si se satisfacen ciertas condiciones. Ambas aproximaciones especifican este tipo de reglas utilizando una notación textual, en la cual se especifica tanto el evento como la acción utilizando pseudo-código orientado a objetos. Ambos métodos difieren entre si ligeramente ya que, por ejemplo, OOHDM utiliza el concepto de post-condición mientras que RUX tiene además en cuenta el comportamiento a lo largo del tiempo de la interfaz. Pero en definitiva, el uso de este tipo de reglas ofrece una gran flexibilidad para definir el comportamiento y, un lenguaje fácilmente comprensible para el analista.

OOH4RIA da un soporte mayor en este aspecto ya que no solo considera el comportamiento de la interfaz como respuesta a un evento, sino también como los distintos *widgets* se comunican entre sí. El modelo de orquestación propuesto permite definir comportamientos más complejos, mediante el encadenamiento de eventos que son enviados a los distintos *widgets,* como consecuencia de los eventos producidos en otros *widgets*.

Por último, cabe destacar que este aspecto no ha sido tratado en detalle en WebML. En el trabajo presentado por [Carughi, Comai *et al.* 2007] se aborda cómo los eventos son definidos y enlazados con los distintos modelos WebML. Sin embargo, las consecuencias de los eventos no definen el comportamiento de la interfaz, sino qué información es recibida/sincronizada desde el servidor o qué nueva tarea tiene que realizar el usuario. Por otro lado, en el marco de UWE no se introduce hasta el momento, el concepto de evento a nivel de interfaz de forma explícita. No obstante, los patrones RIA que se proponen tienen ciertas consecuencias en el comportamiento de la IU.

4) Soporte a patrones habituales en las aplicaciones Web 2.0

A pesar de que no existe un definición ampliamente consensuada para caracterizar formalmente a una aplicación Web 2.0, si que existen una serie

de mecanismos y buenas prácticas que se repiten sistemáticamente en este tipo de aplicaciones. Estos mecanismos tienen como fin simplificar y fomentar la implicación del usuario con la creación, valoración y categorización de los contenidos de la aplicación. Como se ha analizado en el estado del arte, diversos autores han recopilado estos mecanismos en forma de patrones. Un método de Ingeniería Web para el desarrollo de aplicaciones Web 2.0 tiene que, por lo tanto, soportar la definición de la expresividad de estos patrones.

Si bien diversos trabajos avalan la utilidad de dichos patrones, ningún método los incluye explícitamente en el proceso de desarrollo. Además, los trabajos realizados en el ámbito de la Ingeniería Web presentan dos inconvenientes: (1) aunque proponen el uso de modelos para documentar los patrones, no especifican cómo dichos modelos son introducidos en el proceso de desarrollo; (2) han sido definidos en el marco de un método específico cuando, precisamente, el objetivo de un patrón es proporcionar una solución de ámbito global. En conclusión, este aspecto presenta el menor grado de soporte por parte de los métodos de Ingeniería Web analizados.

A modo de resumen de esta sección, para cada aspecto concreto –los cuatro que se acaban de presentar- se ha evaluado el grado de soporte que proporciona el método de Ingeniería Web utilizando la siguiente escala: ningún soporte, soporte bajo, soporte parcial y soporte alto. Los resultados se resumen en la tabla que se muestra a continuación:

Tabla 3.1: Comparativa de los métodos de Ingeniería Web

Método	1. Interacción	2. IU en RIA	3. Eventos	4. Patrones
WebML	Alto	Bajo	Bajo	Parcial
RUX-Method	Bajo	Alto	Alto	Ninguno
OOH4RIA	Alto	Parcial	Alto	Ninguno
OOHDM	Alto	Parcial	Alto	Bajo
UWE	Parcial	Bajo	Bajo	Parcial

La presente tesis doctoral tiene como objetivo proporcionar un soporte alto para los cuatro aspectos planteados a través de las distintas contribucio-

nes. Este hecho convierte al método OOWS 2.0, en el primero en abordar simultáneamente el modelado de la interacción y la interfaz de las aplicaciones Web 2.0 y, en el primero en otorgar especial relevancia a la inclusión, como modelos conceptuales, de patrones de interacción provenientes de aplicaciones Web 2.0.

3.3 Trabajos relacionados en el desarrollo de RIA

A continuación se presentan otras aproximaciones que, aunque no han sido abordadas en el ámbito de un método de Ingeniería Web, también aportan soluciones a la problemática del desarrollo de RIA.

[Dolog & Stage 2007] proponen el modelado de interfaces RIA mediante una extensión basada en UML. A partir de un modelo de tareas, esta aproximación introduce dos conceptos que refinan la semántica de dicho modelo. En primer lugar, se propone la utilización de los llamados *Interaction Spaces:* modelos conceptuales que representan un fragmento de la IU con la cual interactúa el usuario para realizar una tarea. Cada *Interaction Space* define una transición entre tareas como consecuencia de la interacción del usuario. Además, se encuentran conectados con las clases del dominio que son utilizadas en la interacción.

En segundo lugar, se utilizan diagramas de estado UML para modelar la transición entre tareas, en donde cada estado representa la recuperación de un fragmento de información y, cada transición de estado representa un cambio en la tarea a realizar como consecuencia de una interacción con el usuario. Por consiguiente, las diferentes tareas son enlazadas a un estado de dicho diagrama. Puesto que un *Interaction Space* se encuentra también enlazado a una tarea/estado, las transiciones de estado representan las dependencias funcionales entre los *Interaction Spaces*. Así se garantiza que la IU siempre se encuentra en un estado consistente y en sincronización con respecto a la información del SI.

La principal limitación de esta propuesta es que solo resuelve un aspecto muy específico de las interfaces RIA: la reacción de la interfaz como respues-

ta a los eventos y su sincronización con la información. Además, el concepto de *Interaction Space* en sí no es novedoso, debido a que otro tipo de modelos conceptuales se han utilizado con el mismo fin en el ámbito de la Ingeniería Web. Por otro lado, los autores justifican el uso de diagramas de estados por el único hecho de formar parte del estándar UML.

En el ámbito del desarrollo de IU mediante modelos conceptuales, el lenguaje USIXML [Limbourg & Vanderdonckt 2004] también ha sido extendido para proporcionar soporte a la generación de interfaces RIA. Los detalles específicos sobre este lenguaje ya han sido tratados en la sección 3.1. En el trabajo realizado por [Ruiz 2007], se introducen dos extensiones fundamentales para la generación de RIA en USIXML: (1) la definición de un conjunto de tareas genéricas que son habitualmente utilizadas por las interfaces RIA y, su introducción en el modelo de tareas de USIXML; (2) la definición de una serie de modelos de interfaz concretos para distintas tecnologías RIA (XAML, OpenLaszlo y FLEX), junto con las transformaciones XSLT para generarlos a partir de una especificación abstracta en USIXML.

Por un lado, aunque las tareas propuestas generalmente se implementan en la parte cliente de la interfaz, esta circunstancia no siempre tiene porqué ser cierta en función de la aplicación a desarrollar. Al mismo tiempo, el hecho de introducir dicha información arquitectónica, aunque sea de forma implícita, en el modelo de tareas no es una solución adecuada ya que dicho modelo se presupone independiente de la tecnología. Por otro lado, la definición de un modelo para cada tecnología RIA garantiza la expresividad necesaria para la generación de código. Sin embargo, la elección del lenguaje XSLT para la definición de este tipo de transformaciones no es adecuada, porque son difíciles de mantener si la transformación no se produce entre documentos XML. Este último hecho dificulta la viabilidad de esta aproximación.

3.4 Trabajos relacionados con la definición de Patrones para la Web 2.0

Como es conocido, el concepto de patrón fue introducido por Alexander [Alexander 1979], en el ámbito de la arquitectura, para destacar las soluciones que podrían aplicarse de forma recurrente en el diseño de nuevos edificios. Esta noción de patrón fue ampliamente adoptada con posterioridad por la comunidad de la Ingeniería del Software. En particular, en el trabajo presentado por Gamma [Gamma, Helm *et al.* 1995], el concepto de patrón se aplica para catalogar las soluciones a una serie de problemas recurrentes en el desarrollo de software orientado a objetos.

Sin embargo, el concepto de patrón no solo se ha limitado al diseño de la funcionalidad, sino que también es posible encontrar catálogos de patrones en el ámbito del desarrollo de IU. Trabajos relevantes en este sentido son los catálogos de patrones propuestos tanto por [Tidwell 2005] como por [Van Wellie 2000]. Ambos catálogos abordan los diversos problemas que se producen, recurrentemente, en el diseño de interfaces de usuario proporcionando soluciones en forma de patrones. Estos patrones se describen de manera eminentemente textual destacando, en este caso, qué aspecto de la usabilidad contribuyen a mejorar. Aunque estos patrones fueron en un primer momento definidos mediante el análisis de las llamadas interfaces de escritorio, con la llegada de las RIA y la posibilidad de crear interfaces más complejas en la Web, muchos de ellos son actualmente aplicables en el marco del desarrollo de aplicaciones Web 2.0. Su principal inconveniente es que no han sido definidos mediante modelos conceptuales, por lo que su inclusión en los métodos DSDM no es trivial.

Como ya se ha puesto de manifiesto a lo largo de la presente tesis, es difícil acotar qué convierte a una aplicación en parte de la Web 2.0. Sin embargo, aunque no es sencillo dar una definición formal, si que es posible detectar una serie de elementos recurrentes en este tipo de aplicaciones. La gran mayoría de aplicaciones Web 2.0 sociales están construidas alrededor de una serie de mecanismos, que simplifican y enfatizan la interacción del usuario con la aplicación. Por ejemplo, la realización de comentarios y valoraciones de los contenidos de un sitio Web con un aprendizaje nulo. Reduciendo la complejidad de la interacción se consigue que los usuarios se impliquen ac-

tivamente en aportar *feedback* y nuevos contenidos a la aplicación. Estos mecanismos son, en gran en medida, los responsables del éxito de este tipo de aplicaciones y han definido, en parte, un "estándar" implícito sobre aquello que una aplicación Web 2.0 tiene que ofrecer: simplicidad en la interacción con el usuario. Dado que podemos afirmar que la aplicaciones Web 2.0 están construidas alrededor de una serie de buenas prácticas, una aproximación interesante es la de recopilar estas prácticas en forma de patrones.

Siguiendo esta línea de razonamiento, se han definido repositorios de patrones enfocados a la Web 2.0 que son accesibles de forma *on-line*. La principal diferencia con respecto a los catálogos de Gamma, Tidwell o Welie, es que estos repositorios han sido creados a partir de aplicaciones Web 2.0, por lo que no solo abordan aspectos relacionados con la IU, sino que también definen buenas prácticas para enfatizar la interacción con el usuario. El ejemplo más relevante es el catálogo propuesto por [Yahoo 2008] (*Yahoo Design Pattern Library*). La principal contribución de este catálogo es que todos los patrones se han aplicado en aplicaciones reales desarrolladas por *Yahoo*. Por consiguiente, han sido ampliamente validados y es posible asimismo obtener las herramientas, *frameworks* y, en algunos casos, el código utilizado para implementar el patrón.

Este repositorio también fue el primero en incluir un conjunto de patrones denominados sociales, los cuales tienen un enfoque distinto a los patrones de interfaz o tecnológicos. Estos patrones se centran en cómo fomentar la participación en la comunidad construida alrededor de la aplicación y, en mejorar la implicación del usuario en la creación de contenidos. En cierta manera, dichos patrones capturan la esencia de la perspectiva social de las llamadas aplicaciones Web 2.0. En concreto, esta librería incluye un total de quince patrones de carácter social. Para cada patrón, la solución es presentada mediante una plantilla que resume el problema tratado, el razonamiento de porqué el patrón es útil y los escenarios en los que es conveniente aplicarlo.

Una extensión a la librería propuesta por *Yahoo*, ha sido propuesta por [Crumlish & Malone 2009] en la cual se introduce el concepto de *Social Pattern*. Según los autores un *Social Pattern* es un bloque que es utilizado para construir una experiencia social en la Web. Siguiendo esta definición, los au-

tores recogen un lista exhaustiva de más de cincuenta *Social Patterns*, que detallan las prácticas sociales habituales en las aplicaciones Web 2.0 más populares. El principal inconveniente del trabajo es que los patrones son brevemente descritos de forma textual, no aportando información de cómo llevarlos a la práctica. Además, muchos de estos patrones no pueden considerarse como problemas recurrentes, sino como muy específicos de una aplicación social determinada.

Otro repositorio interesante se encuentra en el sitio Web *UIPatterns* [Toxboe 2009]. Aunque este repositorio se centra fundamentalmente en el diseño de interfaces Web, también incluye un conjunto de siete patrones que pueden ser considerados como *Social Patterns* utilizando la terminología de *Crumlish* y *Malone*. Estos patrones son descritos utilizando una plantilla textual, en la cual se destaca cuando debe usarse el patrón y, un razonamiento tanto sobre la solución propuesta como de las ventajas que aporta el uso del patrón. Cabe destacar que los patrones son creados de forma colaborativa, puesto que el repositorio de patrones es en realidad un *wiki*. Por lo tanto, el repositorio en sí mismo también es un ejemplo de aplicación Web 2.0.

Los catálogos o repositorios descritos hasta ahora especifican los patrones, aplicables en la Web 2.0, de manera textual. Este hecho implica que el patrón tiene que ser interpretado a la hora de aplicarlo en el desarrollo de software. En consecuencia, no hay una formalización explícita de cómo aplicar el patrón y el resultado que cabe esperar. En el marco de la Ingeniería Web, la gran mayoría de métodos se han basado en el uso de modelos conceptuales para describir la aplicación Web. Un paso previo a la introducción de los patrones en dichos métodos es la formalización de los mismos como patrones.

Debido a que la inclusión de patrones en el ámbito del DSDM es un tema relevante, diversos trabajos han abordado como llevar acabo dicha inclusión. En primer lugar, [Levendovszky, Lengyel *et al.* 2009] justifican la necesidad de incluir la solución representada por un patrón a nivel de metamodelado. El fin no es otro que crear lenguajes específicos de dominio (DSL) formados por las mejores prácticas detectadas. En concreto, los autores proponen la inclusión en el mismo metamodelo que representa el DSL, la propia representación conceptual del patrón. Para satisfacer dicho objetivo, pre-

sentan el concepto de instanciación parcial que habilita la creación de instancias de los patrones gracias a la relajación de las restricciones, por ejemplo de cardinalidad entre las distintas entidades, a nivel de metamodelo que se tienen que cumplir. La principal contribución de este trabajo es la de establecer tanto un marco teórico como los constructores de modelado necesarios, para llevar a cabo dicha tarea. Aunque el trabajo no se enfoca directamente al ámbito de la Web 2.0, al tratarse de una aproximación genérica a nivel de metamodelado muchas de las ideas son perfectamente válidas.

Son varios los trabajos que se han centrado en soportar un patrón, que ha adquirido una especial relevancia en el ámbito de la Web 2.0: la creación de *tags* o etiquetas. El uso de etiquetas se ha convertido en un mecanismo sencillo para que el usuario clasifique el contenido disponible en la Web. Diversos autores han comprobado que los beneficios de explotar dicha información son muy amplios. En este sentido, el trabajo de [Farrell, Lau *et al.* 2009] analiza las etiquetas asociadas entre los distintos miembros del directorio de una empresa. El resultado es que el mecanismo de etiquetado permite la detección, de forma casi espontánea, de las distintas comunidades de usuarios de la organización. Por lo tanto, este trabajo documenta las ventajas de utilizar este patrón en el contexto de la organización de un SI empresarial.

La principal diferenciación de la presente tesis respecto a las librerías de patrones presentadas radica en la formalización del patrón. En estas librerías, la solución es descrita textualmente siguiendo una estructura predefinida, extraída de la literatura clásica sobre patrones de diseño. Opcionalmente, con el objetivo de hacer más comprensible la solución propuesta, se ofrecen descripciones más detalladas, capturas de pantalla o incluso el código fuente de una implementación. A pesar de todo, en última instancia es el desarrollador quién interpreta la solución descrita y quién decide la implementación específica según sus necesidades. De esta manera, se delega completamente la interpretación del patrón al desarrollador en vez de a los analistas. Esta circunstancia conlleva el riesgo que el patrón no represente en la aplicación final, aquello que se pretendía en la fase de análisis.

3.5 Conclusiones

El presente capítulo ha analizado los trabajos realizados en las distintas áreas que se relacionan con las contribuciones de la tesis doctoral. En primer lugar, se han detallado los trabajos provenientes de la comunidad IPO para el modelado de la interacción. La conclusión principal, obtenida del análisis de estos trabajos, es el énfasis en establecer dos niveles de abstracción. Siguiendo dicha conclusión, nuestra propuesta de modelado de la interacción se sitúa en el denominado nivel abstracto. Por otra parte, la propuesta para el soporte de interfaces RIA se sitúa en el nivel concreto. Además, estos trabajos introducen una serie de modelos para abordar el modelado de la interacción. Sus principales inconvenientes son su falta de expresividad para modelar la interacción de las aplicaciones Web 2.0 y, no considerar la integración con modelos que representen la funcionalidad. Ambas carencias son abordadas por el Modelo de Interacción Abstracto propuesto en esta tesis doctoral.

Respecto a los métodos de Ingeniería Web, se han analizado los trabajos que han introducido extensiones en este ámbito. En los cinco métodos analizados, se destaca un claro interés por el modelado de las características específicas del desarrollo de RIA. Desde el punto de vista de la tesis doctoral, son interesantes aquellas extensiones para modelar interfaces de usuario avanzadas. En este sentido, los modelos conceptuales propuestos son, en líneas generales, suficientes para abordar el desarrollo de aplicaciones Web 2.0. Sin embargo, otros aspectos, considerados también de especial relevancia en el marco de la tesis doctoral, no son tratados. Por ejemplo, a pesar del creciente interés por los patrones en la Web 2.0, ningún método aborda en detalle, salvo WebML, cómo introducir este tipo de expresividad a nivel conceptual. La principal diferenciación de esta tesis reside en combinar ambos aspectos, interfaces RIA y patrones Web 2.0, en el marco del método OOWS 2.0.

Capítulo 4

Especificación del Modelo de Interacción Abstracto

El objetivo del presente capítulo es detallar el Modelo de Interacción Abstracto que es introducido en el método OOWS 2.0. La motivación de este modelo es la de detallar de forma precisa la interacción con el usuario. Con dicho objetivo, en la sección 4.1 introducimos qué se entiende por interacción en el contexto de esta tesis doctoral. Además, se justifica el porqué modelar la interacción. A continuación se evalúa en la sección 4.2, que aspectos de la interacción son tratados actualmente por los modelos conceptuales de OO-Method y OOWS. La sección 4.3 detalla un Modelo de Interacción Abstracto que soporta la definición de la interacción para aplicaciones Web 2.0. Por último, en la sección 4.4, se describen las conclusiones.

4.1 Modelado de la interacción

En primer lugar, entendemos en el contexto de esta tesis la interacción tal y como es definida en [Dix, Finlay *et al.* 1997]: la comunicación entre el usuario y el sistema software para realizar una tarea. A partir de esta definición de interacción, definimos por lo tanto la interfaz de usuario (IU) como una im-

plementación tecnológica para llevar a cabo la interacción. De esta definición se desprende que una IU es una proyección concreta, de entre todas las posibles, para llevar a cabo la interacción que se desea representar. Por lo tanto, aunque existe una estrecha relación entre los conceptos de interacción e IU ambos no son equivalentes. La principal diferencia es que la interacción define el flujo de comunicación entre el usuario y el sistema, mientras que la interfaz define los componente tecnológicos que son utilizados para llevar a cabo dicha interacción. Por ejemplo, en el marco de una aplicación Web 2.0, una interacción habitual es la recuperación de la información más reciente sobre nuestros contactos. Esta información es mostrada mediante una interfaz Web formada por una serie de componentes gráficos de interfaz o *widgets*. Sin embargo, si la misma interacción se realizase desde un dispositivo móvil, la interfaz proporcionada sería completamente distinta.

Puesto que la presente tesis se basa en un método de Ingeniería Web dirigido por modelos, el siguiente paso es definir que se entiende por modelar la interacción. Definimos el modelado de la interacción como la representación de la interacción mediante modelos conceptuales que capturan, de forma precisa y no ambigua, el proceso interactivo. La misión principal de dichos modelos de interacción es su utilización posterior, para generar una IU que soporte adecuadamente la interacción definida.

Los métodos de Ingeniería Web no han puesto demasiado énfasis en el modelado de la interacción proponiendo modelos que se centran en definir, principalmente, la funcionalidad y la interfaz del SI. Las aplicaciones Web 2.0 han puesto de manifiesto que proporcionar una interacción eficaz y sencilla con el usuario es un elemento fundamental. En consecuencia, la falta de mecanismos precisos para modelar la interacción es todavía más grave en el ámbito de la Web 2.0, en donde se busca enfatizar la implicación del usuario con la aplicación. Bajo nuestro punto de vista, creemos que el soporte al modelado de la interacción es un requisito esencial, para que un método de Ingeniería Web cubra, eficientemente, la producción de este tipo de aplicaciones.

El modelado de la interacción es un tema de investigación que ya ha sido tratado por varios autores en el ámbito del modelado de la IU, pero para el cual no se ha consensuado todavía una solución. A la hora de afrontar esta

problemática se distinguen dos aproximaciones. En primer lugar, se encuentran una serie de métodos y herramientas que se especializan en modelar la interacción en un dominio concreto, como puede ser la Web o los dispositivos móviles. Estos métodos adaptan modelos y notaciones tradicionales de la Ingeniería del Software, para soportar la definición de la interacción teniendo en cuenta las características concretas del dominio tecnológico. En el ámbito de las aplicaciones Web son abundantes este tipo de propuestas. Claros ejemplos son los modelos de presentación y navegación de varios métodos de Ingeniería Web como UWE [Koch. 2000], WebML [Ceri, Fraternali *et al.* 2000] o WSDM [Troyer, Casteleyn *et al.* 2008] entre otros. Lamentablemente, esta especialización para un dominio concreto, implica que la expresividad de los modelos ha sido definida teniendo en mente la interacción soportada por un conjunto específico de tecnologías. En el caso de los métodos de Ingeniería Web, los modelos han sido definidos con el objetivo de generar interfaces Web basadas en el lenguaje HTML. Como consecuencia, la migración de estos modelos al ámbito de la Web 2.0 no es trivial, pues implica encontrar equivalencias entre lenguaje tecnológicos que no contemplan, en principio, la misma expresividad a nivel de interacción.

Otra aproximación, muy habitual en el ámbito de la Interacción Persona-Ordenador (IPO), es la definición de un modelo abstracto totalmente independiente de la tecnología que es utilizado para producir la interfaz. Ejemplos de trabajos en esta línea son los lenguajes de modelado de USIXML [Limbourg & Vanderdonckt 2004], TERESA [Mori, Paterno *et al.* 2004] o UMLi [Silva & Paton 2003]. Sin embargo, los modelos abstractos que proponen son muy genéricos y, tienen un nivel de abstracción muy elevado con respecto a la IU que se desea definir. La ventaja es que los modelos son sencillos de definir y de comprender para el analista, pero apenas aportan información para abordar un proceso que tenga como objetivo la generación de la IU final. Para solventar este problema, proponen un segundo nivel de abstracción, denominado concreto, que introduce un nuevo conjunto de modelos que expresen las características específicas de la tecnología. A pesar de ello, las transformaciones a dichos modelos concretos no son triviales puesto que el *gap* semántico respecto al modelo abstracto es muy elevado. Además, mientras que las entidades y la formalización de los modelos abstractos ha sido ampliamente discutida en la literatura, la especificación de los modelos concretos no se ha tratado con el mismo nivel de profundidad.

Independientemente de la aproximación escogida, un inconveniente común en estos trabajos es el hecho de tratar la interacción como parte del modelado de la IU. Debido a esto es habitual que se traten en los mismos modelos aspectos visuales de la interfaz, como son el tamaño o la disposición de los componentes gráficos, junto con aspectos relacionados con la interacción, como son los eventos que inician la comunicación con el SI o la información a introducir. La consecuencia directa de esta aproximación son modelos más complejos que, en principio, tienen que ser definidos tanto por los analistas como por los diseñadores gráficos.

En el contexto del centro de investigación donde se ha desarrollado esta tesis doctoral, también se ha tratado la problemática del modelado de la interacción pero desde la perspectiva de la IU. Para cubrir los aspectos relacionados con la interfaz de usuario se ha definido, en el marco de OO-Method, un Modelo de Presentación [Molina 2003]. Dicho modelo ha sido implementado industrialmente en la herramienta *OLIVANOVA* permitiendo generar interfaces de usuario, en distintas tecnologías como .NET o *Java*, partiendo de la misma especificación conceptual. Este hecho ha permitido validar el Modelo de Presentación OO-Method mediante varias aplicaciones industriales.

El proceso de transformación de esta herramienta está definido de tal forma que, a partir de una primitiva conceptual del Modelo de Presentación, se genera un componente de interfaz predefinido. Esta aproximación implica que cada primitiva del Modelo de Presentación está relacionada con una única implementación o componente gráfico de interfaz. Para seleccionar qué componente de interfaz es el más adecuado para implementar el modelo, se han seguido estándares desde el punto de vista de la usabilidad. A pesar de ello, mediante la retroalimentación de los usuarios que han utilizado las aplicaciones generadas con *OLIVANOVA*, se ha comprobado que la implementación escogida no es en todos los casos la más adecuada. En aplicaciones de escritorio, en donde las IU son más homogéneas y siguen unas reglas de estilo claramente establecidas (disposición de las entradas del menú, formularios de entrada de datos estándar, etc.), este problema no es tan grave puesto que el proceso de transformación utilizado enfatiza esta homogeneidad. Sin embargo, en los entornos Web existe una falta de estandarización a la hora de definir la IU. Es por lo tanto en estos entornos, donde las aplicaciones gene-

radas padecen de problemas de usabilidad, debido a que se han trasladado las mismas IU que eran válidas en aplicaciones de escritorio a la Web.

Con la finalidad de suplir las carencias detectadas para modelar la interacción en ambientes Web, el método OOWS propone al igual que OO-Method un Modelo de Presentación pero orientado a interfaces Web. Por lo tanto, OOWS sigue la misma aproximación que otros métodos de Ingeniería Web, definiendo un modelo que soporta la interacción de forma específica para ambientes Web. Para la validación de la propuesta, se implementaron varias interfaces Web que eran generadas mediante un *framework* escrito en el lenguaje PHP. No obstante, a la hora de llevar a la práctica esta aproximación en el entorno de la Web 2.0, se detectó un grave inconveniente: como consecuencia de la evolución tecnológica experimentada, el Modelo de Presentación de OOWS no era lo suficientemente expresivo. Aunque en principio una evolución tecnológica no tiene por que afectar a la interacción, ya que esta se sitúa a un nivel más abstracto, las aplicaciones Web 2.0 han empezado a incorporar interacciones con el usuario que tradicionalmente estaban restringidas a aplicaciones de escritorio. Al proporcionar tecnologías más avanzadas y flexibles al desarrollador, las nuevas aplicaciones Web han evolucionado considerablemente desde el punto de la interacción. Esta evolución ha sido tal que la línea entre una aplicación Web y una de escritorio comienza a ser difusa, siendo la única diferencia la máquina en la cual se ejecutan.

El Modelo de Presentación OOWS, al igual que modelos equivalentes en otros métodos de Ingeniería Web, fue definido pensado en las interacciones con el usuario que podía ofrecer el lenguaje HTML. Obviamente, si una interacción no podía ser soportada por la tecnología en la cual iba a ser expresada, carecía de sentido introducir complejidad adicional a nivel conceptual. Las nuevas interacciones son únicamente novedosas desde el punto de vista de la Web porque, fuera de este ámbito, sí que han sido tratadas por diversos trabajos de la comunidad IPO.

Para dar soporte a la producción de aplicaciones Web 2.0 creemos que es fundamental definir la interacción de forma precisa. Con este objetivo en mente, se propone y desarrolla en el presente capítulo el Modelo de Interacción Abstracto para OOWS 2.0. Dicho modelo es abstracto porque no está enlazado con una tecnología específica. Así, el analista obvia los aspectos pu-

ramente tecnológicos y se centra en la definición del proceso interactivo. Este modelo abstracto no ha sido definido desde cero, sino que se toman los modelos conceptuales de OO-Method y OOWS como punto de partida. Esta decisión se ha considerado oportuna, ya que un modelo de interacción para la Web 2.0 puede considerarse como la evolución de un modelo de interacción para la "Web 1.0", al cual se le introducen las interacciones más habituales de los entornos de escritorio. Además, de este modo se aprovechan las experiencias obtenidas en la utilización de dichos métodos y sus herramientas asociadas. Por lo tanto, este nuevo modelo de interacción es introducido en el método OOWS 2.0 sustituyendo los modelos actuales de OOWS.

Debido a que el Modelo de Interacción Abstracto ha sido definido independientemente de aspectos tecnológicos, los mismos conceptos pueden ser trasladados a otros métodos de Ingeniería Web. No obstante, cabe señalar que los aspectos tecnológicos tienen que ser convenientemente tratados a fin de conseguir IU de calidad. Dicha problemática es abordada en el capítulo 5 en el cual se define un Modelo de Interfaz RIA. De esta forma, es posible generar interacciones en función de la tecnología destino.

4.2 Soporte a la interacción en OO-Method y OOWS

En esta sección se analiza qué primitivas conceptuales de los modelos de OOWS y OO-Method son reutilizadas para la definición del nuevo Modelo de Interacción Abstracto. Este análisis tiene dos objetivos principales: (1) definir de forma precisa las diferencias entre los modelos de ambos métodos; (2) seleccionar qué primitivas conceptuales son más adecuadas desde el punto de vista de la interacción. Ambos modelos han sido analizados con detalle en las secciones 2.3 y 2.4. Para realizar el análisis, se han clasificado las primitivas conceptuales en base a las tres interacciones genéricas más habituales: recuperación de información, ejecución de servicios y navegación.

4.2.1 Recuperación de la información

Esta interacción es la más relevante puesto que el objetivo esencial de la Web no es otro que el de proporcionar información. Sin embargo, la recuperación

de información no se limita únicamente a enviar datos al usuario. También debe incluir mecanismos de acceso que faciliten el acceso a la información deseada.

Tanto OOWS como OO-Method comparten el mismo modelo para describir qué información almacena la aplicación: el Modelo de Objetos. Básicamente, el Modelo de Objetos es una versión extendida del Diagrama de Clases de UML. También ambos métodos coinciden en el hecho de utilizar este modelo como base para definir la recuperación de la información. Por un lado, en el Modelo de Navegación de OOWS, la recuperación de información se realiza a través de las *Abstract Information Units* (AIU). Esta primitiva conceptual representa una vista sobre dicho Modelo de Objetos, es decir, que atributos de un conjunto de clases forman la información. De esta forma, únicamente se recupera la información de las instancias que forman parte de la vista.

Por otro lado, la misma interacción se puede modelar en el Modelo de Presentación OO-Method mediante tres posibles Unidades de Interacción (UI): Población, Instancia o Maestro-Detalle. El uso de cada una depende de la cardinalidad de la información que se pretende recuperar (una instancia o varias) o, si existe una relación conceptual en la información a mostrar (entre la clase maestra y la clase detalle). La relación entre las UI de OO-Method y el Modelo de Objetos se realiza mediante la asociación con una clase, en el caso de las UI de Población e Instancia, o con dos clases, en el caso de la UI Maestro-Detalle. A diferencia de OOWS, salvo que se utilice el patrón elemental *Display Set*, se muestran y recuperan todos los atributos de las clases.

Como se observa, el concepto de AIU es muy flexible puesto que su expresividad engloba las posibilidades de recuperación de la información, que en OO-Method se realizan mediante tres primitivas conceptuales. Esta opción es más adecuada puesto que simplifica el modelo a construir. Además desde el punto de vista de la interacción, las tres UI representan la misma comunicación con el sistema variando, únicamente, la cardinalidad de la información recuperada. Cabe señalar que las UI guardan una estrecha relación con la interfaz que se muestra al usuario. Sin embargo, la UI de Población se representa en OOWS aplicando el patrón de presentación "tabular-

horizontal", mientras que la UI de Instancia es equivalente a la aplicación del patrón de presentación "registro". Además, una ventaja de las AIU de OOWS es que la expresividad que representa una UI Maestro-Detalle se genera, automáticamente, si una clase directora se relaciona con multiplicidad n con una clase complementaria de la AIU.

Tanto en las AIU de OOWS como en las respectivas UI de OO-Method, la información a mostrar se compone por defecto de todas las instancias. En consecuencia, es necesario el uso de mecanismos de acceso para restringir la población. El uso de dichos mecanismos de acceso por parte del usuario se considera parte de la interacción. En OO-Method dichos mecanismos se representan mediante el uso de un conjunto de patrones elementales: *Filter, Order Criteria* y *Display Set* .

El patrón *Filter* tiene su equivalente directo en OOWS en el Modelo de Navegación. La diferencia entre ambas primitivas conceptuales reside en como se definen las fórmulas de filtrado. En OO-Method, los filtros se definen mediante una fórmula bien formada en forma de expresión lógica abierta que puede contener varias variables sin valor explícito. El usuario tiene que introducir, por lo tanto en tiempo de ejecución, el valor de las variables de la fórmula para definir una expresión lógica cerrada que se evalúa para cada instancia de la población. Por otra parte, en OOWS se diferencian dos tipos de filtros. En primer lugar, se dispone de los filtros dinámicos con una semántica idéntica a la de OO-Method, con la salvedad que se definen mediante una única variable. En segundo lugar, se introduce el concepto de filtro estático que está compuesto por una expresión lógica cerrada. En este caso, el usuario no introduce ningún valor y únicamente decide si desea activar o no el filtro. Además, en OOWS, también es posible definir un filtro estático que se active, de manera automática siempre que se recupere la información, para establecer una condición de filtrado poblacional.

Otro mecanismo de acceso, que también se encuentra en ambos modelos, es el patrón elemental *Order Criteria*. Ambos modelos soportan este tipo de ordenación en base a uno o varios atributos de la AIU o de la UI respectivamente. La única diferencia es que en OO-Method se definen ordenaciones basadas simultáneamente en varios atributos, mientras que en OOWS la or-

denación únicamente actúa, ascendentemente o descendentemente, sobre los valores de un atributo.

Cabe destacar un mecanismo de acceso presente solo en OOWS como es el índice. Este mecanismo define una clasificación de las instancias a mostrar en la AIU en función del valor de un atributo. Cuando el usuario selecciona un valor del índice, únicamente las instancias que cumplen con dicho valor son mostradas. El mecanismo de índice no se encuentra soportado de forma implícita por el Modelo de Presentación de OO-Method, si bien es cierto que mediante un filtro definido con una variable objeto-valuada es posible simular un comportamiento similar.

En el Modelo de Presentación OO-Method, cuando se especifica el patrón *Filter* u *Order Criteria* a una población, la información no es recuperada hasta que el usuario activa dichos patrones. Para modificar este comportamiento y cargar todas las instancias, se usa el patrón *Population Preload*. Este patrón, además, se utiliza para recuperar los posibles valores de argumento de tipo objeto. En OOWS dicho patrón no es necesario ya que la población siempre es recuperada, independientemente, de la definición o no de un mecanismo de acceso. Para evitar la carga excesiva de información se usan las vistas de resultados. Una vista de resultados especifica un resumen de la información recuperada (un subconjunto de los atributos de la AIU) a través de la aplicación de un filtro o índice. En consecuencia, se le muestra al usuario dicha información y, cuando se selecciona la instancia deseada, se muestra al completo la vista definida en la AIU. El objetivo principal de la vista de resultados es mejorar la usabilidad cuando un gran número de instancias son recuperadas.

Ambos modelos también disponen de un mecanismo para dividir la información en bloques de una cardinalidad dada. OOWS extiende esta posibilidad definiendo como el usuario accede a esos bloques: de forma secuencial o de forma aleatoria mediante la selección del bloque de instancias a mostrar. También se permite en OOWS que el usuario defina la cardinalidad de los bloques en tiempo de ejecución a diferencia de OO-Method, en dónde es el analista quien fija qué número de instancias muestra por defecto una UI de población. Respecto al patrón de *Display Set* de OO-Method,

OOWS permite definir la misma semántica mediante los distintos atributos de las clases de la AIU.

4.2.2 Ejecución de servicios

La interacción para la ejecución de servicios tiene que proporcionar al usuario un diálogo o formulario, para la introducción de la información necesaria en la ejecución de la funcionalidad. En OO-Method dicho diálogo se modela mediante una UI de Servicio asociada a un servicio del Modelo de Objetos. En el Modelo de Navegación de OOWS, los servicios que pueden ser ejecutados se definen en las clases que forman las vistas de las AIU estableciendo, así, la relación con el método del Modelo de Objetos.

Es aquí donde acaban las similitudes entre ambos modelos. En OO-Method al crear una UI de Servicio se asume, por defecto, la aplicación del patrón *Entry* sobre cada uno de los argumentos del servicio. Al estar definido cada argumento mediante este patrón, es posible aplicar, en primer lugar, máscaras de edición para restringir el conjunto de valores posibles de un argumento. También es posible otros patrones con la misma función tales como *Defined List, Argument Dependency* o *State Recovery*. La combinación de estos patrones restringe la introducción de valores y, evita la gran mayoría de las situaciones en las cuales el usuario introduce un valor erróneo. También es posible establecer valores iniciales para un argumento o, definir reglas de dependencia entre los valores de dos argumentos de tal forma que cuando uno es introducido, el otro sea modificado en función de una fórmula lógica.

Por otra parte, el Modelo de Navegación OOWS no proporciona, de forma directa, el soporte necesario para evitar la introducción errónea de valores. Es el compilador de modelos OOWS quién genera un formulario por defecto acorde con los argumentos del servicio. Por lo tanto, no es posible definir en la fase de modelado expresividad adicional para restringir los valores de los argumentos. Si bien el compilador "deduce" a partir del tipo de datos del argumento que valores son aceptables o no, la expresividad proporcionada resulta insuficiente en la práctica. En este aspecto, la UI de Servicio definida en OO-Method ofrece una mayor expresividad.

En OO-Method, el patrón *Editable Display Set* también define una interacción muy habitual, como es la edición de la información que ha sido recuperada previamente. Usando dicho patrón, el usuario puede editar la información sobre la misma UI mejorando, por tanto, la usabilidad. Realmente, este patrón se considera mixto porque se aplica simultáneamente tanto a una UI de Población como a una de Servicio. En OOWS no se contempla esta interacción.

Respecto al patrón *Outbound Arguments* de OO-Method, su utilidad reside en proporcionar retroalimentación adicional al usuario cuando un servicio ha sido ejecutado. Este patrón recoge los valores de retorno del servicio y permite seleccionar cuales son mostrados al usuario. En OOWS no se define, de forma concreta, qué tiene que mostrarse al usuario cuando un servicio es ejecutado. La implementación presentada en [Valverde 2007] asume que se muestra un mensaje con el resultado de la ejecución y, el valor de todos los argumentos de retorno si los hubiera.

4.2.3 Acceso a la aplicación y navegación

La interfaz de una aplicación puede describirse como un conjunto de nodos interrelacionados, en donde cada uno abstrae un requisito de interacción del usuario con el sistema. Tomando esta definición, llamamos navegación a la transición desde un nodo a otro. Por lo tanto, el acceso a la aplicación de un usuario se define en base a los nodos accesibles, ya sea de forma directa o mediante el uso de la navegación. Estableciendo paralelismos entre ambos métodos, en OO-Method estos nodos son las UI mientras que en OOWS son los Contextos de Navegación.

Se diferencian dos tipos de navegaciones en función de la alcanzabilidad de los nodos por parte del usuario. Aquellas navegaciones que se encuentran disponibles independientemente del nodo en el cual se encuentra, las denominamos de acceso al sistema. Por otro lado, aquellas que únicamente se pueden activar en uno o varios nodos de la interfaz, es decir siguiendo un camino predefinido, reciben el nombre de navegación por secuencia. El tratamiento que se hace de ambos tipos de navegaciones difiere sustancialmente a la hora de especificarlas en ambos métodos.

Para cubrir la navegación de acceso al sistema, en el Modelo de Presentación OO-Method se utiliza el Árbol de Jerarquía de Acciones (AJA). Esta primitiva conceptual define una estructura en forma de árbol en la cual, los nodos intermedios definen agrupaciones de la funcionalidad (por orden alfabético, tarea, significado, etc.) para facilitar su localización mientras que las hojas hacen referencia a una UI. El usuario tiene acceso directo a las UI definidas en los nodos hoja, mientras que el resto de UI del Modelo de Presentación solo son accesibles mediante navegaciones. La implementación habitual de un AJA en la interfaz final es el menú principal de la aplicación.

La primitiva equivalente al AJA en OOWS es el Mapa de Navegación. Este se representa usando un grafo dirigido, en el cual los nodos son los Contextos de Navegación y los arcos son los enlaces (o vínculos) de navegación. Se distinguen tres tipos de Contextos de Navegación: (1) exploración, si siempre está disponible para el usuario, (2) secuencia, si el contexto solo es alcanzable a través de la navegación desde otro contexto y (3) inicio, el cual es único en el Mapa de Navegación ya que define el contexto inicial que es mostrado al usuario al entrar en la aplicación. Al igual que el AJA, los Contextos de Navegación de inicio y de exploración dan lugar en la interfaz, a un menú que siempre permanece visible para el usuario.

La principal diferencia entre ambas propuestas es que mientras OOWS define la relación de navegación entre distintos contextos, es decir las navegaciones por secuencia, en el AJA únicamente se define la navegación de primer nivel o de exploración. El acceso a otras UI destino no está expresado implícitamente con lo cual, el AJA no proporciona una la visión general del sistema, aunque puede ser derivada posteriormente.

Además, OOWS permite definir relaciones de especialización basadas en el Diagrama de Usuarios, el cual indica para cada usuario su Mapa de Navegación. En consecuencia, es posible definir entre dos usuarios una relación de herencia de tal forma que el usuario hijo herede el Mapa de Navegación completo. Este mecanismo facilita la definición de mapas de navegacionales que, únicamente, difieren en pocos contextos respecto al Mapa de Navegación del usuario raíz. Aunque el reuso de la misma UI entre distintos AJA si que es posible, en OO-Method no se dispone de un mecanismo de herencia similar.

Para modelar la navegación de secuencia en el Modelo de Presentación de OO-Method se utiliza el patrón elemental de navegación. Este patrón se asocia a una relación estructural a la cual pertenezca la clase que define una UI de Población o Instancia. Como destino se selecciona una UI de Instancia o Maestro-Detalle cuya clase también pertenezca a la relación. Por lo tanto, cuando se activa la navegación seleccionando un objeto en la UI origen, se navega a la UI destino mostrando la información que se encuentra relacionada con el objeto seleccionado.

Dos patrones adicionales permiten definir interacciones de navegación con comportamientos diferentes. En primer lugar, el patrón *Navigation Filtering* define una fórmula de filtrado que es activada en la UI destino. Mediante este patrón se consigue, que dependiendo de la navegación a través de la cual se alcance una UI, su información varíe. En segundo lugar, el patrón *Navigational Condition* define una navegación que no se asocia a una relación estructural, sino a la ejecución de un servicio. En este caso, cuando finaliza la ejecución, se navega a una UI distinta teniendo en cuenta la evaluación de una serie de condiciones. El uso de este patrón configura qué UI es la más adecuada a mostrar al usuario teniendo en cuenta los resultados de un servicio.

OOWS, aunque también define la navegación en función de relaciones estructurales entre clases, aporta una mayor expresividad a esta interacción. La diferencia fundamental entre ambas propuestas es que OOWS proporciona distintas alternativas a la hora de iniciar la navegación: mediante un atributo de la información recuperada (navegación por objeto), mediante una relación estructural (navegación por relación) o mediante la ejecución de un servicio (enlace de servicio). El primer caso se asemeja a la navegación definida por el patrón elemental de OO-Method salvo que, en este caso, un atributo del objeto actúa como enlace en la interfaz para iniciar la navegación. Sin embargo, en la navegación por relación, se tiene en cuenta no un único objeto, sino el conjunto de objetos que se obtienen a través de una relación de cardinalidad *n*. De esta forma, se recupera simultáneamente información de varios objetos en el contexto destino. Por último, el enlace de servicio es una versión simplificada del patrón *Navigational Condition* puesto que, en este caso, no se definen condiciones adicionales a cumplir. En todos los casos al realizar la navegación también se tiene en cuenta la información referente

al objeto, relación o servicio, sobre el cual se aplicó la navegación. Esta información se encuentra disponible en el contexto destino para ser utilizada.

A modo de resumen, se muestra en las siguientes tablas una comparativa entre ambos métodos, destacando las primitivas conceptuales que son relevantes para modelar la interacción. Así pues, para cada primitiva conceptual del Modelo de Presentación de OO-Method, se menciona su equivalencia más próxima en los modelos OOWS detallando, brevemente, las posibles diferencias. En el caso que no exista una primitiva conceptual que exprese la misma semántica, se detalla si es posible soportar dicha expresividad de manera más o menos fiel mediante otros mecanismos. La misma comparativa es también realizada en el sentido inverso para detectar que expresividad de OOWS, no está contemplada en OO-Method.

Tabla 4.1: Comparativa primitivas conceptuales OO-Method/OOWS

OO-Method	Equivalente en OOWS
H.A.T Tree	Contextos de Navegación de inicio y exploración del Mapa de Navegación
Population IU	AIU con el patrón de presentación tabular horizontal
Instance IU	AIU con el patrón de presentación registro
Service IU	Sin equivalente. En la fase de generación de código se proporciona un formulario por cada método de la AIU
Master-Detail IU	AIU formada por dos clases relacionadas con cardinalidad n más el patrón de presentación maestro-detalle
Entry	Sin equivalente. En la fase de generación de código se tiene en cuenta el tipo de datos a introducir
Display Set	Conjunto de atributos de las clases de la AIU
Editable Display Set	Sin equivalente
Defined Selection	Sin equivalente
Dependency Rule	Sin equivalente
Complementary Info	Clases complementarias de la AIU

OO-Method	Equivalente en OOWS
Filter	Filtro dinámico o estático pero sobre un único atributo
State Recovery	Sin equivalente. Puede consultarse información sobre el objeto que inicia la navegación o sobre el cual se ejecuta un servicio
Order Criterion	Patrón de presentación de ordenación sobre un atributo de una AIU
Tree View	AIU con el patrón de presentación árbol
Conditional Navigation	Enlace de servicio pero sin condiciones
Navigational filtering	Sin equivalente
Navigations	Relaciones de contexto definidas sobre una relación entre dos clases de una AIU
Actions	Servicios en las vistas que forman parte de una AIU
Outbound Arguments	Sin equivalente. Los argumentos de retorno de los servicios son mostrados en un mensaje por defecto
Population Preload	Sin equivalente. La información siempre se carga haya o no definido un mecanismo de acceso

Tabla 4.2: Comparativa primitivas conceptuales OOWS/OO-Method

OOWS	Equivalente en OO-Method
Mapa de usuarios	Sin equivalente. Los agentes juegan el rol de autentificación
Mapa Navegacional	Árbol AJA
Contexto Navegacional Exploración	Nodos hoja del árbol AJA
Contexto Navegacional de Secuencia	Sin equivalente. UI que no se encuentran en el árbol AJA y pueden ser alcanzadas mediante el patrón de Navegación
AIU	UIs de Población, Registro y Maestro-Detalle y patrón de Acciones
Clase directora	Clase sobre la cual se define una UI

OOWS	Equivalente en OO-Method
Clase Complementaria	Clase "detalle" en una UI Maestro-Detalle y patrón de Información Complementaria
Relación de contexto de objeto	Patrón elemental de Navegación
Relación de contexto relacional	Sin equivalente
Enlace de servicio	Navegación condicional
Filtro dinámico	Filtro con variables
Filtro estático	Filtro sin variables
Filtrado poblacional	Sin equivalente
Índice	Sin equivalente. Filtro definido con una variable cuyos posibles valores sean los del índice
Vista de resultados	Sin equivalente. Definición de una UI que represente la vista de resultados y con una navegación hacia una UI el resto de información
Condición de Navegación	Sin equivalente
Navegación con cambio de usuario	Sin equivalente
Patrón de ordenación	Patrón elemental de ordenación
Patrón de disposición	UI de Población, Registro y Maestro-Detalle. Sin equivalente para la disposición tabular horizontal, texto y árbol
Patrón de paginación	Sin equivalente. Propiedad para definir en máximo número de instancias en la UI de población
Patrón de orden de aparición	Sin equivalente

Como se concluye del presente análisis, ambos modelos tienen sus puntos débiles y sus puntos fuertes. El objetivo del Modelo de Interacción Abstracto es recopilar las primitivas conceptuales más adecuadas de ambos. En líneas generales, se ha optado por utilizar el Diagrama de Usuarios y el concepto de Mapa de Navegación presentado por OOWS. La razón principal es que ofrece una visión global del sistema y es una notación más comprensible

para el analista. También se reutiliza de OOWS el concepto de AIU al proporcionar una notación visual y más compacta, que la equivalente de OO-Method. Respecto a la ejecución de servicios, se opta por el uso de la UI de Servicio que proporciona la expresividad necesaria para esta tarea. Con el objetivo de soportar el resto de primitivas conceptuales, se ha introducido en el modelo un nivel de patrones elementales al igual que en OO-Method. Mediante el concepto de patrón auxiliar de interacción, se recopilan tanto los patrones de OO-Method como mecanismo de acceso de OOWS, como los índices o las vistas de resultados, que resultan útiles. De esta manera, se proporciona una visión más estructurada del modelo.

4.3 Definición del Modelo de Interacción Abstracto

El Modelo de Interacción Abstracto es una parte fundamental del método OOWS 2.0 puesto que define el proceso interactivo con el usuario. Este modelo de interacción actúa como nexo entre el sistema de información y el usuario para ofrecerle, tanto la información como la funcionalidad descrita en el Modelo de Objetos. Se hace especial hincapié en la palabra abstracto ya que, aunque en el marco de la presente tesis es utilizado para definir la interacción con aplicaciones Web 2.0, el modelo en sí es totalmente independiente de la plataforma y la tecnología de desarrollo utilizada. La semántica del modelo propuesto junto con las primitivas conceptuales que lo componen se detalla a continuación. Para ilustrar mejor los distintos conceptos se ha utilizado una aplicación Web 2.0 para la gestión de la actividad científica. Este ejemplo ilustrativo puede consultarse en el Anexo A.1.

4.3.1 El Diagrama de Usuarios y el Mapa de Interacción

Definimos el Modelo de Interacción Abstracto como un modelo conceptual [Kühne 2004] cuyas primitivas capturan la interacción [Dix, Finlay *et al.* 1997] usuario-sistema sin contemplar los aspectos tecnológicos relacionados con esta interacción. Al tratarse de un modelo conceptual, su formalización se basa en un metamodelo compuesto del conjunto de primitivas conceptuales que forman dicho modelo. Este metamodelo ha sido definido mediante

Ecore [Budinsky, Steinberg *et al.* 2009], el lenguaje de metamodelado de la plataforma Eclipse basado en el lenguaje *MOF Essential;* un subconjunto del estándar MOF [OMG 2006] propuesto por la OMG. Mediante este metamodelo, se construyen los modelos que cumplen formalmente con el Modelo de Interacción Abstracto propuesto. El Modelo de Interacción Abstracto complementa los modelos de OO-Method y, en particular, se relaciona directamente con el Modelo de Objetos que describe las entidades del SI.

La primitiva conceptual principal que articula el modelo son los Contextos de Interacción o *Interaction Contexts* (IC). Un Contexto de Interacción describe una tarea (por ejemplo realizar una compra, mostrar los clientes dados de alta, introducir los datos personales, etc.) que el usuario realiza con el SI. Desde el punto de vista conceptual, un IC es un contenedor de las distintas primitivas conceptuales que describen la interacción asociada a una tarea. De esta manera, en un punto concreto de la interacción con la aplicación, el usuario solo se encuentra en un IC. Introduciendo este concepto, el proceso de interacción con la aplicación se define como la sucesión de los distintos IC que el usuario utiliza.

La definición del acceso a los diferentes IC se realiza a través de dos vistas que se complementan entre sí: el Diagrama de Usuarios y el Mapa de Interacción. Por un lado, el Diagrama de Usuarios es utilizado para caracterizar los distintos tipos de usuarios que tienen acceso a la aplicación. Por otro lado, un Mapa de Interacción asocia a cada usuario los IC a los cuales tiene acceso.

Un IC se compone de al menos una Unidad de Interacción Abstracta o *Abstract Interaction Unit* (AIU) que abstrae, una interacción que se asocia con uno o varios elementos del Modelo de Objetos (clases, atributos, relaciones y servicios). En otras palabras, las AIU son las encargadas de mostrar la información y la funcionalidad al usuario definida en el Modelo de Objetos. La semántica de esta primitiva conceptual es similar a la utilizada en el Modelo de Navegación de OOWS. Obviamente, únicamente definiendo la interacción en base a relaciones con el Modelo de Objetos, no es posible capturar toda la expresividad necesaria. Para introducir esta semántica adicional, se extiende la expresividad de las AIU mediante un conjunto predefinido de Patrones de Interacción Auxiliares o *Auxiliary Interaction Patterns* (AIP).

El Diagrama de Usuarios representa los distintos conjuntos de usuarios que pueden interactuar con el sistema. Por conjunto de usuarios, entendemos a aquellos agentes humanos externos que comparten la misma interacción con el sistema de información. Por ejemplo, en el ejemplo ilustrativo escogido, los tres tipos de usuario son el usuario "Investigador", que representa a aquellas personas que se han dado de alta en el sistema para llevar a cabo la gestión de su investigación, el usuario "Anónimo", que únicamente puede consultar información y, por último, el usuario "Administrador" encargado de mantener la información de la aplicación.

Cada usuario del Diagrama de Usuarios se encuentra relacionado con un agente perteneciente al Modelo de Objetos de OO-Method. En OO-Method para cada agente, el cual es representado mediante una clase, se establece la visibilidad de atributos, servicios y relaciones que tiene sobre el resto de clases del sistema. De esta forma, el analista restringe la interacción con el sistema a los distintos agentes indicando a qué información tienen acceso, qué servicios pueden ejecutar y qué vista del Modelo de Presentación representa su interfaz. Para la distinguir a los distintos agentes se define una interacción implícita que consiste en la autenticación en la aplicación. De esta forma, cada vez que un usuario quiere acceder al sistema tiene que introducir su identificador de usuario, su contraseña y seleccionar a que agente pertenece.

El Diagrama de Usuarios aquí presentado es compatible con la definición de agente de OO-Method. Cada usuario es asociado igualmente a una clase del Modelo de Objetos a la cual denominamos *User Class*. No obstante, ampliamos la expresividad de los agentes de OO-Method introduciendo dos extensiones:

- En primer lugar, a cada usuario además de asociarle una *User Class*, que representa a un agente de OO-Method, se le asocia una vista de usuario (*User View*) sobre una o varias clases relacionadas con la *User Class*. Los atributos de las clases de esta vista representan la información del usuario que se encuentra disponible a la hora de modelar la interacción. En la Fig. 4.1 se muestra la definición de esta vista para el usuario *Researcher*. Asociando dicha *User Class* con una vista compuesta por la clase "*Affiliation*", siempre que el usuario se

identifique en el sistema, estará disponible en todos los IC tanto la información relevante de su perfil como la información correspondiente a su afiliación. Esta información puede ser utilizada en los distintos IC con el objetivo de configurar la interacción. Cabe destacar, que únicamente está disponible la información que representan los atributos presentes en la vista de usuario, es decir, no toda aquella que representa la clase.

- Se introduce el concepto de usuario anónimo (*Anonymous User*) para representar el acceso al sistema sin necesidad de identificarse. Al no ser necesaria la interacción de autenticación, tampoco tiene sentido asociar al usuario anónimo una clase del Modelo de Objetos debido a que, realmente, no pertenece al sistema. De la misma forma, la definición de vistas de usuario está fuera de lugar puesto que no es posible almacenar información de un usuario anónimo y, después, distinguir a que usuario pertenecía.

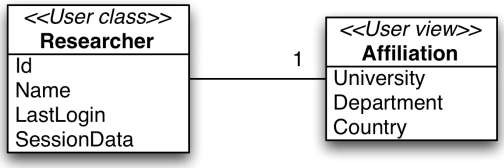


Fig. 4.1: *User Class* para el agente "*Researcher*"

Para cada tipo de usuario, se asocia el conjunto de IC al cual tiene acceso definiendo su Mapa de Interacción. Este mapa representa una vista global de la interacción del usuario con el sistema y estructura el acceso a los distintos IC. De esta forma el Mapa de Interacción define las transiciones permitidas entre los distintos IC. Los IC se clasifican en el Mapa de Interacción, desde el punto de vista de su accesibilidad, como de primer nivel (*First level)* o de secuencia (*Sequence)*. Los IC de primer nivel representan aquellas interacciones que el usuario realiza en cualquier momento mientras que los IC de secuencia, son aquéllos que únicamente son accesibles a través de otro IC. Este último tipo de IC es útil para definir secuencias de interacciones, asociadas a tareas que tienen que ser realizadas en un orden preestablecido. La Fig. 4.2 muestra el Mapa de Interacción simplificado para el usuario *Researcher* en el cual se aprecian los conceptos introducidos. Para representar gráficamente al

usuario, utilizamos la primitiva conceptual "actor" perteneciente a los Casos de Uso de UML. Por otra parte, un paquetes UML, estereotipado mediante el tipo de accesibilidad, representa un IC. Los distintos paquetes que forman el Mapa de Interacción se encuentran relacionados mediante flechas de tal forma que, aquéllos que reciben una flecha desde el usuario son considerados de primer nivel. Por otra parte, aquellos IC que únicamente son alcanzables desde otro IC se consideran de secuencia. Analizando el Mapa de Interacción de la Fig. 4.2, para que el usuario *Researcher* realice la interacción "*Modify Profile*", previamente tiene que consultar su perfil personal mediante la interacción "*View my Profile*". Para mantener la compatibilidad con los agentes de OO-Method a todo usuario, exceptuando al anónimo, se le asocia un IC de primer nivel denominado *Login* el cual, implícitamente, representa la interacción de autenticación para acceder a la interfaz de la aplicación.

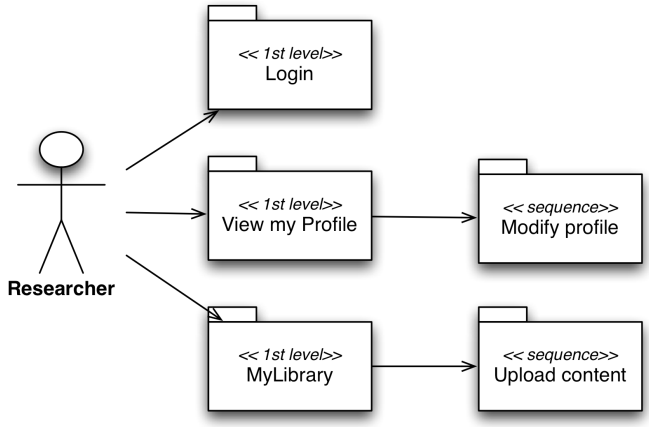


Fig. 4.2: Mapa de Interacción para el usuario *Researcher*

Si bien el Mapa de Interacción es un modelo necesario, no se define de forma implícita por el analista. Este modelo se deriva a partir de la definición de los distintos IC que forman el sistema. Cuando el analista define un IC, únicamente tiene que especificar si es un contexto de primer nivel o no. El resto de IC se consideran, por lo tanto, de secuencia y son accesibles mediante interacciones de navegación desde un IC de primer nivel u otro de secuencia. A partir de esta información, la herramienta de modelado tiene que analizar las distintas navegaciones definidas construyendo, automáticamente, el

Mapa de Interacción. Mediante la construcción del Mapa de Interacción, es posible verificar que no existan IC que no son alcanzables a través de alguna navegación.

El Diagrama de Usuarios también define la jerarquía entre los usuarios a través de relaciones de herencia. Haciendo uso de la generalización de actores, tal y como se define en UML, un usuario puede heredar el Mapa de Interacción de un usuario base. De esta forma, el usuario hijo tiene acceso tanto a las interacciones del usuario base, como a aquéllas que defina su propio Mapa de Interacción. Para facilitar la herencia es posible definir usuarios estructurales. Estos usuarios no tienen acceso al sistema y se utilizan para agrupar IC, facilitando la definición de Mapas de Navegación complejos. Este tipo de usuario es útil cuando el mapa perteneciente a dos usuarios únicamente se diferencia en uno o pocos IC. Ya que no se soporta la redefinición de IC, la solución consiste en definir un usuario estructural cuyo Mapa de Interacción aglutine las IC comunes a ambos. En consecuencia, ambos usuarios heredan de dicho usuario estructural y añaden los IC específicos de cada uno.

4.3.2 Unidades de Interacción Abstractas

Las Unidades de Interacción Abstractas o *Abstract Interaction Units* (AIU) son las primitivas conceptuales atómicas de las cuales se componen los IC. En el contexto del método OOWS 2.0, la interacción se produce con respecto a los modelos OO-Method que describen el SI, es decir, el Modelo Dinámico, de Objetos y Funcional. Sin embargo, es realmente el Modelo de Objetos quién describe la interfaz con el SI. Teniendo en cuenta la expresividad ofrecida por dicho modelo, se distinguen dos interacciones posibles con este modelo: (1) la recuperación de la información sobre el estado de uno o varios objetos; (2) la ejecución de un servicio sobre un objeto determinado. Se observa que, aunque desde la perspectiva del usuario la interacción es más compleja, en última instancia, desde la perspectiva del SI ambas interacciones son su única interfaz con el exterior. Estas dos posibles interacciones dan lugar, respectivamente, a los dos tipos de AIU que son definidos en el modelo: *Population AIU* y *Service AIU*.

La *Population AIU* abstrae una interacción muy habitual como es la recuperación de información estructurada. Siguiendo una aproximación similar a la de OOWS, esta AIU especifica la recuperación de información a través de una vista sobre un conjunto de clases pertenecientes al Modelo de Objetos. La información recuperada es, por tanto, el estado de los objetos sobre los que se defina la vista. La especificación de una *Population AIU* (ver Fig. 4.3) se compone de relaciones con primitivas conceptuales del Modelo de Objetos. En primer lugar, se define una *Manager Class* (clases UML estereotipadas con la palabra *Manager)* o clase directora y, opcionalmente, varias *Complementary Class* (estereotipadas con la palabra *Complementary*), que sean alcanzables desde la *Manager Class* a través de un conjunto de asociaciones presentes en el Modelo de Objetos. A partir de la *Manager Class* definida, se recuperan todas las instancias de la misma junto con las instancias de las *Complementary Classes* que se encuentren relacionadas. Los atributos mostrados son aquéllos definidos en la vista.

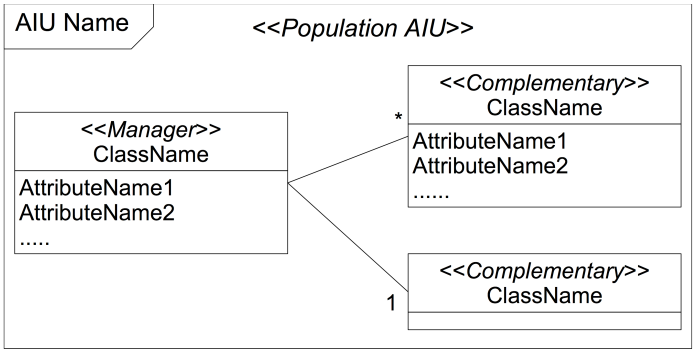


Fig. 4.3: Notación para la especificación de la *Population AIU*

La Fig. 4.4 muestra una ejemplo de la definición de una *Population AIU* para la recuperación de un listado de artículos. La *Manager Class* recupera el título, año y un enlace al documento para cada artículo. Además, como información complementaria, se recupera el tipo de publicación, el lugar de publicación y un conjunto de palabras clave. Se observa que la relación con la clase *Keywords* es de multiplicidad *n*, lo cual implica que para cada artículo se recuperan varias instancias de dicha clase. Las otras dos relaciones con multiplicidad 1 implican la recuperación de una, y solo una, instancia relacionada.

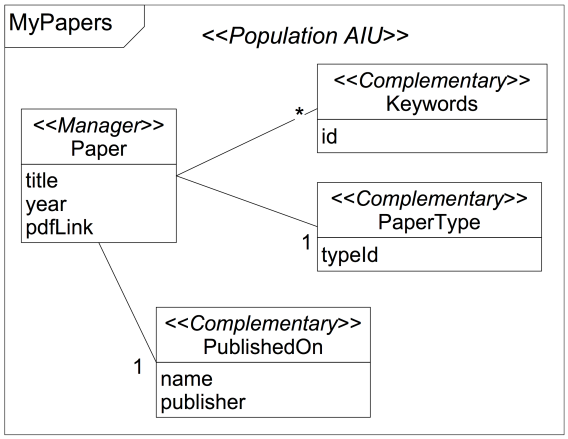


Fig. 4.4: Ejemplo de una *Population AIU*

Generalmente, después de la recuperación de la información, la interacción más habitual en un SI es la ejecución de servicios que proporcionan la funcionalidad. El objetivo de la Unidad de Interacción Abstracta de Servicio o *Service AIU* es el de abstraer la ejecución de un servicio de la lógica de negocio, proporcionando al usuario un diálogo para introducir sus argumentos e iniciar la ejecución. Para especificar una *Service AIU*, se ha definido una plantilla textual en vez de una notación gráfica tal y como muestra la Tabla 4.3. Este tipo de notación resulta más práctica y legible, que la utilización de modelos gráficos basados en UML. Además, las notaciones basadas en plantillas han sido utilizadas en diversos métodos de modelado para la especificación de funcionalidad, como los presentados por [Valderas 2008] o por [Escalona 2004].

Una *Service AIU* se define mediante una relación con un servicio perteneciente a una clase del Modelo de Objetos. Dicho servicio se compone a su vez de una serie de argumentos. Por lo tanto, la *Service AIU* se define además en base a un conjunto de argumentos de entrada (*Input Argument)*, que el usuario introduce, y un conjunto de argumentos de salida (*Output Argument*), que informan al usuario del resultado de la ejecución. Cada argumento es definido en una nueva fila de la plantilla introduciendo su identificador en el Modelo de Objetos (*id)* y su alias en el modelo de interacción. Si el alias se omite se asume que se usa el *id* como alias.

A cada *Input Argument*, es posible asociarle un valor por defecto (*default value*) que sea compatible con el tipo de datos del argumento (*type*). Para la definición de los valores por defecto, pueden también utilizarse valores que se encuentren disponibles en el IC al cual pertenece la AIU. Este valor por defecto es una sugerencia al usuario que puede ser modificado, cuando se produzca la interacción. La ejecución de una operación del Modelo de Objetos implica la inserción de un valor para cada argumento. Sin embargo, es posible definir argumentos sobre los cuales el usuario no interactúa. Un argumento constante (*Constant Argument*) implica que no está disponible desde el punto de vista de la interacción o, en otras palabras el usuario no tiene que introducirlo dado que su valor nunca varia. Todo *Constant Argument* tiene que poseer, salvo que acepte el valor nulo en su definición, un valor por defecto que es el utilizado en cada ejecución.

La definición de los *Output Arguments* es similar salvo que, en este caso, puede especificarse la visibilidad (*visibility)* del argumento. De esta forma, se controla qué atributos son mostrados o no al usuario. Por último, también es posible definir los distintos errores que se producen debidos a una ejecución incorrecta. Estos errores pueden estar asociados a una condición que, en caso de cumplirse, muestre el mensaje correspondiente.

Tabla 4.3: Plantilla para la especificación de una *Service AIU*

Service AIU	Name				
Service	class name.operation id				
Input Arguments	id	alias	type	default	constant
	string	<string>	datatype : <cardinality> or {list_of_values}	value	bool
Output Arguments	id	alias	type	visibility	
	string	<string>	datatype : <cardinality>	bool	
Errors	Condition		message		
	<if expression>		String		

La Tabla 4.4 muestra una *Service AIU* con la especificación de un servicio para añadir un nuevo artículo a la librería del investigador. Esta *Service AIU* se encuentra enlazado con el servicio *Upload* de la clase *Paper* y consta de

cuatro argumentos. El identificador del artículo no tiene que ser introducido por el usuario, es un valor que se incrementa automáticamente en el sistema, por este motivo se define como *Constant*. Se definen además dos argumentos de salida: un enlace al archivo del artículo, una vez ha sido añadido al sistema, y un código de error. Este último código no es mostrado al usuario, pero sus valores son utilizados en la sección *Errors* para definir los distintos mensajes de error que se producen: que el fichero no exista o que no haya podido ser almacenado.

Tabla 4.4: Ejemplo de una *Service AIU*

Service AIU	Upload a new Paper				
Service	Paper.upload				
Input Arguments	id	alias	type	default	constant
	paperId	Id	autoincrement	1	true
	title		string:128	'paper title'	false
	type	publication type	string: 32	'article'	false
	file	file to upload	string: 128		false
Output Arguments	id	alias	type	visibility	
	link	link to paper	string: 256	true	
	errorId		Int	false	
Error Feedback	condition		message		
	If errorId == -1		'The file cannot be found'		
	If errorId == -2		'File cannot be uploaded. Please try again'		

4.3.3 Patrones Auxiliares de Interacción

Las AIU expresan las interacciones básicas con el SI, sin embargo, desde el punto de vista de la aplicación las interacciones son más complejas. Analizando diversas aplicaciones se ha detectado que existe un conjunto acotado de interacciones que extienden la expresividad abstraída por las AIU. La diferencia fundamental es que estas interacciones no se producen de forma directa con el SI sino, o bien con la información recuperada, o bien en el marco de la ejecución de un servicio. Aunque estas interacciones pueden ser incluidas en la semántica de las AIU, para facilitar la labor del analista y definir un

modelo que sea extensible de forma incremental, se ha decidido encapsularlas en forma de Patrones Auxiliares de Interacción o AIP (*Abstract Interaction Patterns*). Se denominan patrones de interacción, porque parten de una solución de interacción que se repite habitualmente, y auxiliares, porque dependen de una AIU para su especificación.

A diferencia de los patrones a nivel de diseño introducidos en trabajos como los de [Tidwell 2005] o [Van Wellie 2000], que definen los patrones a nivel de implementación (Espacio de la Solución), en el marco de esta tesis se definen los patrones a nivel conceptual (Espacio del Problema). Esta estrategia es, además, coherente con los distintos niveles del Modelo de Presentación OO-Method actual, el cual hace uso de un concepto similar mediante los llamados patrones elementales. Por lo tanto, utilizando esta aproximación se han extendido, en algunos casos, patrones elementales que ya existían en el Modelo de Presentación. Para definir los patrones de una manera estructurada y homogénea, se han seguido los principios definidos por [Gamma, Helm *et al.* 1995] y la plantilla propuesta en [Molina 2003]. Según la propuesta de *Gamma*, un patrón debe estar compuesto de cuatro elementos fundamentales:

1. Un nombre que describa el problema a resolver de una forma abstracta.

2. El problema que soluciona el patrón teniendo en cuenta un escenario concreto.

3. La solución que describe los elementos que forman el patrón y su interrelación entre ellos.

4. Las consecuencias, en forma de ventajas e inconvenientes, de aplicar el patrón.

En función de estos principios, se ha elaborado una plantilla para cada patrón que consta de los siguientes apartados:

- **Nombre**: identificación breve de la interacción que abstrae el patrón.

- **Fundamento**: este apartado describe, de forma breve y concisa, qué interacción modela el patrón, cual es el propósito del mismo y qué problema plantea resolver. Es similar al apartado *Intent* propuesto por *Gamma*.

- **Especificación**: detalla como tiene que usarse a nivel de modelado, el patrón para crear una instancia del mismo y cuales son los parámetros disponibles para el analista. Agrupa los apartados de *Participants* y *Collaborations* presentados en *Gamma*.

- **Semántica**: en este apartado se detalla cual es el comportamiento esperado del patrón cuando es aplicado. Con dicho fin establece la correspondencia entre cada elemento conceptual de patrón y su aportación a la interacción representada.

- **Notación**: indica que notación es utilizada para representar el patrón, es decir, que elemento textual o gráfico representa el patrón a la hora de construir el modelo. Las notaciones de carácter textual se han formalizado utilizando XText [Behrens, Clay *et al.* 2009], lenguaje que se explica con mayor detalle en la sección 5.2.2.

- **Ejemplo**: describe un ejemplo concreto de aplicación del patrón utilizando una aplicación real. En este caso, se utiliza el ejemplo ilustrativo descrito en el Anexo A.1.

- **Metamodelo:** especifica de forma gráfica el patrón de interacción utilizando *Essential MOF* [OMG 2006] como lenguaje de metamodelado. El metamodelo describe los elementos conceptuales que forman el patrón, las propiedades a especificar y las relaciones con las AIU u otros elementos del metamodelo. El equivalente a esta sección según Gamma es el apartado *Structure*.

Esta plantilla es una simplificación de toda la expresividad necesaria ya que el objetivo del patrón, no es el de ser una guía detallada de cómo definir las reglas de producción del código asociado a la interacción. Esencialmente, un AIP define las entidades a nivel de metamodelado necesarias, como tienen que ser utilizadas a través de una notación y la semántica esperada cuando se aplica el patrón. A la hora de definir el conjunto de AIP, se ha tenido

en cuenta como se abordó el modelado de interacción en aproximaciones anteriores. Por lo tanto, los patrones aquí definidos amplían y mejoran primitivas conceptuales previas, extendiendo su expresividad donde ha sido necesario. Por razones de legibilidad, se han omitido los metamodelos correspondientes, pero pueden consultarse en el Anexo A.2. La Fig. 4.5 muestra el conjunto de AIP definidos, junto con sus relaciones con el resto de las primitivas conceptuales del Modelo de Interacción Abstracto.

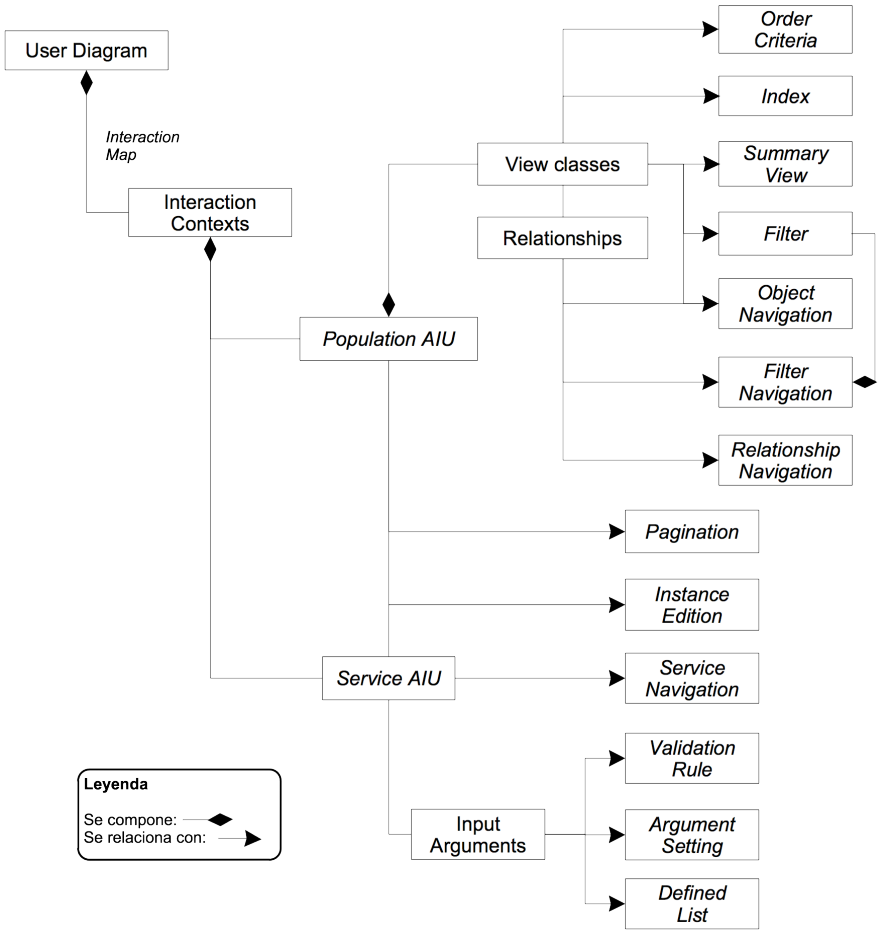


Fig. 4.5: Estructura del Modelo de Interacción Abstracto

4.3.3.1 Filtro (Filter)

Fundamento: Una *Population AIU* recupera, por defecto, todas las instancias de las clases que se hayan incluido en la vista. Si el número de instancias es muy elevado, la cantidad de información puede ser excesiva para que el usuario la consulte de forma cómoda. El *Filter AIP* permite al analista restringir el conjunto de instancias que se muestran cuando se produce la interacción. Con dicho fin, se define una condición de filtrado, mediante una fórmula bien formada, que descarta aquellas instancias que no la cumplen.

Especificación: Un *Filter AIP* se especifica asociándose a una *Population AIU* y a una fórmula que define la condición de filtrado. La fórmula se compone de una sucesión de átomos que tienen que ser evaluados a cierto para que una instancia sea mostrada. Para definir fórmulas complejas los átomos pueden ser concatenados utilizando los operadores lógicos *AND* y *OR*. Un átomo se compone fundamentalmente de un atributo perteneciente a la *Population AIU* correspondiente, un operador y un valor de filtrado.

El conjunto de operadores soportados para establecer la condición de filtrado de un átomo son: *Equal, Greater, GreaterEqual, Lesser, LesserEqual*, para definir comparaciones de orden entre el valor de filtrado y el valor del atributo, y *Like*, para filtrar aquellas instancias en donde el valor de filtrado no se encuentra contenido en el valor del atributo. Respecto al tipo de valores de filtrado disponibles, es posible utilizar valores constantes de un tipo de datos específico, por ejemplo un valor entero, valores que se encuentren disponibles en el ámbito del Contexto de Interacción o valores dinámicos, representados como variables de entrada, que son introducidos por el usuario. Como atributo opcional a la hora de definir un filtro, es posible definir una descripción que proporcione al usuario una mayor información sobre la misión del mismo.

Semántica: El *Filter AIP* se presenta como un mecanismo que el usuario activa, voluntariamente, cuando desea limitar el número de instancias. Cuando el filtro es activado, se comprueban cada una de los átomos definidos y solo son mostradas, aquellas instancias cuyos atributos satisfacen todos los átomos de la fórmula de filtrado. En función del tipo de valor de filtrado de los átomos, se distinguen dos tipos de filtro que tienen distinto comportamiento:

- **Dinámico**: se define mediante una fórmula abierta, es decir, el valor de filtrado del atributo no es especificado en la fórmula. Es el usuario quien completa la fórmula en tiempo de ejecución introduciendo un valor. Por ejemplo, un filtro dinámico sobre una variable de filtro "años" implica que el usuario introduzca el año de las instancias a visualizar.

- **Estático**: se define como una fórmula cerrada de tal forma que, el usuario no puede cambiar el valor del atributo sobre el cual se filtran las instancias de la población. En el caso de definir un filtro estático sobre el atributo "año", el usuario elige si filtrar o no la población mediante el año proporcionado, por ejemplo si el año es igual a 2000, pero no puede modificar el año por el cual se filtran.

Notación: el *Filter AIP* se define mediante una formula textual utilizando la siguiente sintaxis formalizada mediante XText:

```
FilterFormula:
    'Filter' name=ID description=STRING
    '{' atoms+=FilterAtom (boolOperator atoms+=FilterAtom)* '}';

FilterAtom:
    attribute=ID opFilter (value=ID | variable=STRING)+;

opFilter:
    'Equal' | 'NotEqual' | 'Greater' | 'GreaterEqual' |
    'Lesser' | 'LesserEqual' | 'Like';

boolOperator:
    'AND' | 'OR';
```

Ejemplo: Dado el gran número de artículos científicos presentes en el sistema se desea obtener únicamente aquéllos publicados con posterioridad al año 2000 o, aquéllos que pertenezcan a un autor en concreto. Suponiendo que se ha definido una *Population AIU* para recuperar el conjunto de artículos, en el primer caso, se define un *Filter AIP* cuya fórmula se compone del siguiente átomo: "*PublishedYear GreaterEqual* 2000" donde *PublishedYear* es un atributo de la *Population AIU*. En el segundo caso, utilizaríamos un filtro dinámico con la fórmula: "*name Like v_AuthorName*". En este caso no especificamos el valor de filtrado, sino que definimos una variable de filtrado "v_AuthorName" para que sea el usuario quien introduzca el valor.

4.3.3.2 Índice (Index)

Fundamento: cuando existe una elevada cantidad de información para mostrar al usuario, resulta interesante dividir esa información en varias categorías. El *Index AIP* divide el conjunto de información en base al valor de un atributo en concreto. De esta forma, el usuario selecciona una única categoría o valor del índice que restringe la información mostrada.

Especificación: Un *Index AIP* se especifica sobre una *Population AIU* y un atributo de alguna de las clases que forman la vista de la AIU. El patrón incluye una serie de atributos opcionales como, una descripción del propósito del índice y un mecanismo para mostrar el número de instancias de cada categoría.

Semántica: El *Index AIP* se define como una variante de un filtro estático compuesto de una variable de tipo objeto, cuyos valores posibles de filtrado están previamente definidos en el Modelo de Objetos. Cuando se aplica el *Index AIP* a una *Population AIU*, se muestra un conjunto de enlaces con los distintos valores que posee el atributo sobre el cual se define el índice. Una vez seleccionado un valor por parte de usuario, se muestran únicamente la instancias que cumplen con dicho valor seleccionado. Opcionalmente, al lado de cada enlace se muestra un número que indica cuantas instancias tiene la *Population AIU* en dicha categoría.

Notación: un *Index AIP* se define mediante una anotación UML sobre la clase a la cual pertenece el atributo. En dicha anotación, se incluye el identificador del atributo, la descripción del índice y si se muestra o no el contador de instancias.

Ejemplo: los distintos artículos dados de alta en el sistema son clasificados utilizando un conjunto de palabras clave almacenadas en la clase *Keyword*. Debido al alto número de artículos, una *Population AIU* definida sobre la clase *Paper* necesita de algún mecanismo de acceso. Definiendo un *Index AIP*, tal y como se muestra en la Fig. 4.6, el usuario puede ver únicamente aquellos artículos que correspondan a una categoría.

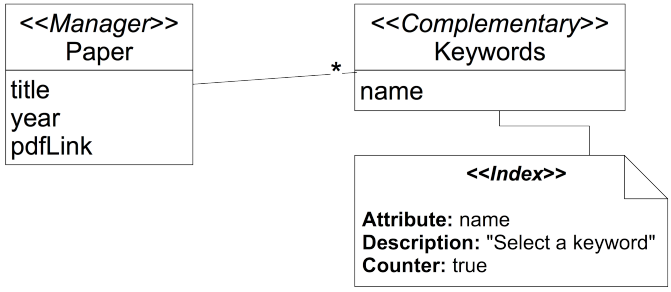


Fig. 4.6: Ejemplo de un *Index AIP*

4.3.3.3 Vista Resumida (Summary View)

Fundamento: Cuando una *Population AIU* está definida mediante varias clases, el conjunto de información mostrado al usuario resulta, generalmente, elevado. El *Summary View AIP* define un subconjunto de dicha información para mostrarla, de forma más sencilla y concisa, al usuario. De esta manera, a partir de dicho resumen, el usuario selecciona una única instancia de la cual se muestra el conjunto completo de información.

Especificación: este AIP se asocia a un conjunto de atributos pertenecientes a la clase directora de una *Population AIU*. Si dicha *Population AIU* tiene definidos un filtro o un índice, también puede asociarse el *Summary View AIP* a estos mecanismos de acceso. Opcionalmente, puede definirse un parámetro denominado umbral de activación, que determina el número de instancias a partir del cual se activa el patrón automáticamente.

Semántica: si el *Summary View AIP* está definido sobre una *Population AIU*, se muestra únicamente la información de los atributos especificados en el patrón, siempre y cuando, se cumpla el umbral de activación o dicho umbral no exista. Para cada instancia, se genera un enlace, u otro mecanismo de selección análogo, que permite al usuario acceder a toda la información definida de la instancia. Si la vista se asocia a un filtro o a un índice la semántica es similar salvo que, en este caso, el *Summary View AIP* se muestra cuando el filtro o índice es activado. La principal ventaja del uso de este patrón es evitar la definición de varias *Population AIU*, para acceder a la misma información con distintos niveles de detalle.

Notación: el *Summary View AIP* se define mediante una plantilla textual en la cual se especifica el conjunto de atributos, entre llaves, junto con un valor numérico para el umbral. La siguiente tabla muestra esta plantilla:

Tabla 4.5: Plantilla para la definición del *Summary View AIP*

Summary View	<text description>
Applied to	<Population AIU> ||<Filter> ||<Index>
Attributes	<{attribute1,…,attributeN}>
Activation Threshold	<integer>

Ejemplo: la información disponible sobre un investigador en la aplicación *MySocialResearch* es muy extensa. Por ejemplo, un Contexto de Interacción que muestre toda la información relevante sobre los investigadores de una red (afiliación, publicaciones, etc.), no resulta usable en la practica. Una solución más adecuada, es realizar un listado por el nombre y apellidos de los investigadores para, posteriormente, acceder a la información detallada. Suponiendo que una *Population AIU,* denominada *Research Info*, muestra dicha información el *Summary View AIP* especificado en la Tabla 4.6 define este comportamiento, siempre y cuando, existan más de cinco investigadores dados de alta en una red.

Tabla 4.6: Ejemplo de un *Summary View AIP*

Summary View	Surname Researcher View
Applied to	Researcher Info AIU
Attributes	{name, surname }
Activation Threshold	5

4.3.3.4 Ordenación (Order Criteria)

Fundamento: La información recuperada a través de una *Population AIU* se devuelve al usuario en el orden que encuentra almacenada, por defecto, acorde con el identificador de la instancia. El *Order Criteria AIP* modifica es-

te comportamiento según el criterio del usuario definiendo enlaces de ordenación, que establecen el orden de las instancias en función del valor de uno o más atributos.

Especificación: Un *Order Criteria AIP* se asocia con una *Population AIU* y con uno o varios atributos definidos en la misma. Para cada atributo, se especifica su prioridad a la hora de realizar la ordenación y el sentido de dicha ordenación: ascendente o descendente. También es posible asociar un *Order Criteria AIP* por defecto, de tal forma que siempre sea aplicado sobre la *Population AIU*.

Semántica: Cuando en una *Population AIU* existe al menos un *Order Criteria AIP* definido, se le proporciona la opción al usuario de activarlo. La ordenación se realiza teniendo en cuenta los valores del atributo de mayor prioridad. En el caso que existan dos atributos con idéntico valor, se utiliza el primer atributo en orden de prioridad y así sucesivamente. El sentido de la ordenación tiene en cuenta los valores de forma alfanumérica, es decir, en primer lugar desde el '0' al '9' y a continuación de la 'A' a la 'Z'.

Notación: Los *Order Criteria AIP* se definen siguiendo la siguiente sintaxis XText:

```
OrderCriteria:
   'OrderCriteria "' description=ID '"' ('default')?
   'by {' order+=AttributeCriteria
   (',' orderAttributes+=AttributeCriteria)*'}';

AttributeCriteria:
   (className=ID'.')?attributeName=ID':'operator=opOrder;

opOrder:
   'ASC'|'DESC';
```

Ejemplo: En una *Population AIU* que muestra el conjunto de artículos de un investigador, resulta adecuado aplicar una ordenación tanto por año como por las palabras claves de cada artículo. El *Order Criteria AIP* para obtener dicha interacción se especifica como:

Order Criteria "Order my paper" by { year:DESC, Keywods.name:ASC }

En donde el atributo *year* pertenece a la clase directora de la AIU, mientras que las distintas palabras clave se ordenan en función del atributo *name* de la clase complementaria *Keywords*.

4.3.3.5 Paginación (Pagination)

Fundamento: Cuando la información a mostrar es elevada, resulta conveniente dividirla en varios subconjuntos en vez de mostrarla toda. El *Pagination AIP* divide la información de una *Population AIU* en conjuntos o páginas de una cardinalidad definida. La interacción que se abstrae mediante este AIP es la selección, por parte del usuario, de una de estas páginas de información para su visualización.

Especificación: El *Pagination AIP* se asocia únicamente con una *Population AIU*. Este patrón posee tres parámetros que especifican, con más detalle, su comportamiento: (1) *Cardinality,* que indica el número de instancias que forma cada grupo de información; (2) *Random*, que activa o no el acceso aleatorio a los grupos de información; (3) *Sequence,* que activa o no el acceso secuencial a cada grupo.

Semántica: La interacción representada por este patrón varia en función de los parámetros establecidos por el analista. En primer lugar, el *Pagination AIP* tiene que tener al menos uno de los dos mecanismos de acceso (*Random* o *Sequence*) activados. Cuando el acceso *Random* está activado, la interacción ofrecida al usuario se representa, mediante un índice numérico que proporciona acceso a un conjunto de instancias de tal forma que, si la cardinalidad es diez, la primera página del índice muestra las diez primeras instancias, la segunda página las diez siguientes y así sucesivamente. El usuario, en este caso, puede acceder directamente a cualquier conjunto. Si el acceso *Sequence* se encuentra activado, la interfaz ofrece al usuario la posibilidad de ver el primer y el último conjunto de instancias, además del siguiente y el anterior al que se encuentra actualmente. Este patrón, es compatible con otros AIP, como el de Filtro u Ordenación, de tal forma que define las páginas en función de la población que restringen estos patrones. La cardinalidad definida por defecto depende de la implementación.

Notación: el *Pagination AIP* se define siguiendo la siguiente sintaxis XText:

```
Pagination:
    'Pagination with' ('cardinality' cardinality=INT ',')?
    (accessType += AccessTypes 'access')?;

AccessTypes:
    'random' | 'sequence';
```

Ejemplo: La siguiente definición de un *Pagination AIP,* "*Pagination with cardinality 20, random access, sequence access*" muestra la información en grupos de 20 instancias. Suponiendo que la *Population AIU* tuviese 200 instancias se generaría un índice de 20 páginas. Para la definición de la librería de artículos de un investigador, este patrón resulta útil en caso que disponga de gran cantidad de material.

4.3.3.6 Selección Enumerada (Defined List)

Fundamento: A la hora de introducir un valor, es habitual que el usuario únicamente elija entre un conjunto acotado de valores o un valor definido previamente en el SI. El *Defined List AIP*, abstrae la interacción mediante la cual se le ofrece al usuario un conjunto de valores, y éste selecciona el que considera adecuado. Gracias a este patrón se evita, en parte, la introducción de valores erróneos.

Especificación: Un *Defined List AIP* se asocia a toda primitiva conceptual que implique la introducción de un valor por parte de un usuario. Por consiguiente, se asocia, o bien a los argumentos de una *Service AIU,* o bien a las variables de un *Filter AIP*. Cada valor se define mediante una dupla [alias, dato], en donde el alias es el valor mostrado al usuario y el dato es el valor real que se introduce en el sistema. El conjunto de valores permitidos se define mediante una lista, que puede ser estática o dinámica. Una lista estática se especifica en tiempo de diseño y consiste en un conjunto de valores que son compatibles con el argumento o variable al cual se aplica el patrón. En cambio, una lista dinámica se define en base a una dupla de atributos pertenecientes al Modelo de Objetos, de modo que, en tiempo de ejecución, se obtiene la información de los valores a introducir.

Semántica: Cuando se aplica el patrón, la interacción proporcionada al usuario pasa de ser una introducción (el usuario escribe el valor) a ser una selección variando, en consecuencia, el conjunto de componentes gráficos de

interfaz compatibles. El conjunto de valores que se muestra al usuario viene dado, por los identificadores de las distintas duplas que definen el conjunto de valores. Si el conjunto ha sido definido mediante una lista estática (*Static List*), los valores posibles son definidos por el analista, y este conjunto no varia a lo largo de la interacción En cambio, si la lista es dinámica (*Dynamic List)*, los valores posibles se recuperan del sistema y, si a causa de algún servicio la población es modificada, el conjunto de valores de la lista también lo hace.

Notación: el *Defined List AIP* se define como una lista de duplas con la siguiente sintaxis: "{ [<*id1,value1*>], …, [<*idN,valueN*>] }", en donde tanto *id* como *value,* hacen referencia a valores estáticos o a atributos pertenecientes, a algún elemento del Modelo de Interacción Abstracto, como por ejemplo, los atributos de las clases de una *Population AIU*.

Ejemplo: Cuando el usuario define su perfil o introduce un nuevo artículo en el sistema, tiene que seleccionar un conjunto de palabras clave adecuadas para su identificación posterior. Ya que solo se aceptan palabras clave previamente introducidas y, estas varían a lo largo del tiempo, se define un *Defined List AIP* de la siguiente forma: "{ [*Keywords.name, Keywords.id*] }". Aplicando este patrón, por ejemplo, sobre el argumento correspondiente del servicio, el usuario visualiza, de forma textual, las distintas palabras claves disponibles, pero el sistema recibe como valor el *id* de la palabra clave.

4.3.3.7 Regla de Validación (Validation Rule)

Fundamento: Habitualmente, un usuario tiene que introducir un valor que corresponda a un tipo de datos específico (entero, texto, etc.) o que cumpla con un formato preestablecido (una dirección de correo o una fecha en un formato determinado). La misión del *Validation Rule AIP* es la de informar al usuario, que el valor introducido no es correcto en base a una expresión lógica.

Especificación: El *Validation Rule AIP* se asocia a toda primitiva conceptual que implique la introducción de un valor por parte del usuario. Por lo tanto, se asocia a los argumentos de una *Service AIU* y a las variables de un *Filter AIP*. Un *Validation Rule AIP* se especifica mediante: (1) una expresión regular o una operación del Modelo de Objetos de tipo *boolean*, que recibe el

valor a introducir como parámetro, (2) un mensaje para informar al usuario qué tipo de valor tiene que introducir y (3) opcionalmente, un mensaje de error que es mostrado si el valor introducido no es correcto. Para la definición de las expresiones regulares, se utiliza la sintaxis PCRE tal y como se especifica en [Hazel 2010].

Semántica: A diferencia del *Defined List AIP*, en este caso, el conjunto de valores no se encuentra acotado de antemano por el analista. Este patrón restringe la interacción, por ejemplo la ejecución de un servicio o la selección de un valor de filtrado, sobre la cual se definen las reglas del tal forma que, hasta que no se cumplen todas, no se habilita su realización. Para guiar al usuario, se muestra un mensaje de ayuda (por ejemplo, al situarse sobre el componente de interfaz para introducir el valor), que indica que tipo de valor se espera que introduzca. Cuando el usuario introduce el valor, inmediatamente se valida si cumple la expresión regular. En el caso que se haya definido una operación para validación, se realiza la invocación a la lógica de negocio con el valor introducido y se tiene en cuenta, si el valor de retorno es cierto. Si el valor introducido no cumple la expresión, automáticamente se muestra un error que informa del error al usuario y le insta a que lo corrija.

Notación: el *Validation Rule AIP* se define utilizando una plantilla textual tal y como muestra la siguiente tabla:

Tabla 4.7: Plantilla para la definición de un *Validation Rule AIP*

Validation Rule	<rule description>
Applied to	<ServiceAIU Argument> ||<Filter variable>
Validation expression	<regular expression> || <operation>
Input tip	<text>
Error message	<text>

Ejemplo: Cuando se introduce una nueva publicación en la librería del investigador, se tiene que introducir la fecha en un formato concreto para que sea comprensible para el sistema. El siguiente *Validation Rule AIP*, aso-

ciado al argumento *publishDate*, obliga a que las fechas cumplan con el formato *dd/mm/yyyy*.

Tabla 4.8: Ejemplo de un *Validation Rule AIP*

Validation Rule	check_year
Applied to	new_publication.publishDate
Validation expression	(0[1-9]|[12][0-9]|3[01])[-/.](0[1-9]|1[012])[-/.](19|20)
Input tip	'Input a date in format dd/mm/yyyy'
Error message	'date format incorrect'

4.3.3.8 Inicialización de Argumento (Argument Setting)

Fundamento: a la hora de ejecutar un servicio, no siempre todos los argumentos tienen que ser introducidos por el usuario, porque el valor de algunos puede derivarse en función de la información ya disponible. El *Argument Setting AIP* define un valor específico para un argumento siempre y cuando, se cumpla una regla. De esta forma, se mejora la usabilidad puesto que el usuario no tiene que introducir valores redundantes y tiene como referencia, un valor inicial que le sirve de ejemplo.

Especificación: este patrón se asocia a cualquier argumento de entrada perteneciente a una *Service AIU*. Respecto a las reglas se distinguen dos tipos: (1) estática, cuando la regla se define en base a un valor predefinido o a un atributo de la instancia, sobre la cual se ejecuta el servicio; (2) dinámica, si el valor del argumento se define en base al valor de otro argumento. Como restricción, el valor seleccionado para inicializar el atributo tiene que ser compatible con el tipo de datos correspondiente.

Semántica: este patrón se aplica, de forma inmediata, cuando el usuario accede a la *Service AIU*. Para la definición de reglas estáticas se utilizan valores fijos (números enteros, texto, etc..) o se consultan los atributos de la instancia, sobre la cual se ejecuta el servicio. Las reglas dinámicas hacen referencia a valores de otros argumentos presentes en la *Service AIU,* de tal manera que cuando uno de dichos valores cambia, también lo hace en consonancia

los argumentos sobre los cuales se aplica el patrón. Ambos tipos de reglas soportan la utilización de operadores aritmético-lógicos, para la definición de inicializaciones complejas.

Cabe destacar, que existen dos posibles problemas a la hora de utilizar este patrón. En primer lugar el patrón es compatible con el *Validation Rule AIP*. Sin embargo, no se realiza una validación en tiempo de análisis para determinar, si la inicialización de argumentos viola sistemáticamente alguna regla de validación. Además, se tienen que evitar la aparición de bucles infinitos cuando existen reglas de inicialización circulares como, por ejemplo, que un argumento A inicialice un argumento B y el argumento B inicialice a su vez al argumento A. En consecuencia, ambos problemas tienen que ser considerados por el analista a la hora de especificar el patrón.

Notación: este patrón se especifica, de manera textual, en una columna de la sección *Input Arguments* perteneciente a la plantilla de definición de una *Service AIU*. La sintaxis para definir la regla de inicialización es en lenguaje XText:

```
ArgumentSetting:
   dyRule=DynamicRule | stRule=ID;

DynamicRule:
   argumentId+=ID (operator=opString stValue=ID)
   | (operator=opInt intValue=INT);

opInt:
   '+' | '-' | '*' | '/';
opString:
   'concat';
```

Ejemplo: cuando el usuario introduce un nuevo artículo científico en el sistema, uno de los campos que tiene que definir es la afiliación de los distintos autores. Con asiduidad, esta información es compartida entre varios de los autores del artículo. Mediante la aplicación de este patrón, es posible inicializar la afiliación de todos los autores a partir de la del primer autor, de modo que el usuario solo tenga que introducir las afiliaciones que son distintas. Suponiendo que el argumento que define la afiliación del autor principal

es "*p_affiliation*", para definir el comportamiento descrito, se especifica un *Argument Setting AIP* como se muestra en la Tabla 4.9.

Tabla 4.9: Ejemplo de un *Argument Setting AIP*

	id	Type	Argument Setting
Input Arguments	p_author_affiliation	string:128	
	p_author2_affiliation	string:128	p_affiliation
	p_author3_affiliation	string:128	p_affiliation

4.3.3.9 Navegación por Objeto (Object Navigation)

Fundamento: Una navegación implica un cambio en el Contexto de Interacción percibido por el usuario. No obstante, es habitual que, simultáneamente a la navegación, en el contexto destino se disponga de información perteneciente al Contexto de Interacción, desde donde se inició la navegación. El *Object Navigation AIP* representa una interacción de navegación que se inicia, cuando el usuario selecciona un objeto específico de una *Population AIU*. Dicha selección inicia una navegación, de tal manera que el contexto destino recibe dicha instancia y muestra, únicamente, la información relacionada con la misma.

Especificación: Una navegación por objeto se define mediante: (1) un atributo (atributo de enlace) perteneciente alguna de las clases complementarias de la *Population AIU*, (2) una relación definida entre dos clases pertenecientes a la *Population AIU* y, (3) otra *Population AIU* perteneciente al contexto destino de la navegación. Adicionalmente, se tiene que cumplir la restricción que la *Population AIU* del contexto destino, esté definida mediante una *Manager Class* que pertenezca a la relación entre las clases de la *Population AIU* del contexto origen.

Semántica: Cuando se muestran todas las instancias de una *Population AIU*, aquellos atributos que tienen asociado un *Object Navigation AIP* son resaltados, con el fin de indicar que son susceptibles a iniciar una navegación, por ejemplo, subrayando sus valores en forma de enlace. Al seleccionar uno de estos valores, se inicia automáticamente la transición al contexto destino, enviando el *oid* de la instancia a la cual pertenece el valor. Una vez en el

Contexto de Interacción destino, la *Population AIU*, con la cual se asoció el *Object Navigation AIP*, tiene como única instancia de la *Manager Class* aquella que se ha recibido a través de la navegación. En consecuencia en este contexto destino, las clases complementarias únicamente muestran información que se relacione con dicha instancia. Este patrón es similar, por tanto, a un filtro estático automático definido sobre el atributo *oid* de la clase directora. Ya que por definición el *oid* es único, la activación de este filtro, a través de la navegación siempre retorna una única instancia perteneciente a la *Manager Class*.

Notación: el *Object Navigation AIP* se define utilizando una relación de la *Population AIU* con una *Complementary Class.* Dicha relación es representada mediante una flecha continua que denota la aplicación del patrón. Como etiqueta de la relación se muestra tanto el atributo de enlace como el contexto destino (entre corchetes), tal y como muestra la siguiente figura:

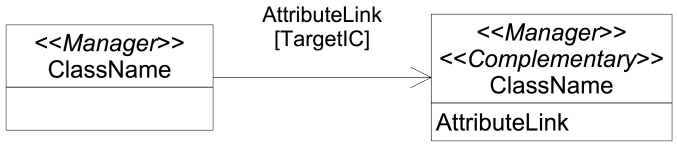


Fig. 4.7: Notación para el *Object Navigation AIP*

Ejemplo: Cuando el usuario se encuentra en su página inicial, puede ver un listado de todas las redes a las cuales pertenece. A fin de consultar el conjunto de investigadores suscritos a una red, se define un *Object Navigation AIP* entre los contextos *Home* y *Social Network Details*. Este último contexto muestra el detalle de una red de investigadores definida en el sistema, siendo su *Manger Class* también *ResearchNet*. La navegación se iniciará cuando el usuario seleccione el atributo *netAlias* desde el Contexto de Interacción *Home*. La Fig. 4.8 muestra el modelo correspondiente a esta interacción en el Contexto de Interacción *Home*.

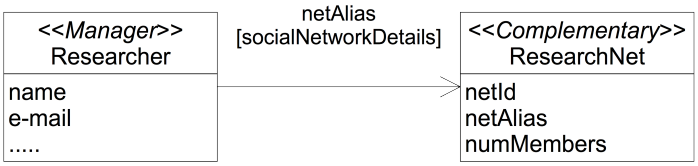


Fig. 4.8: Ejemplo de un *Object Navigation AIP*

4.3.3.10 Navegación Relacional (Relationship Navigation)

Fundamento: Este patrón es una variación del *Object Navigation AIP*, con el fin de obtener la información relacionada con un objeto determinado. El *Relationship Navigation AIP* define una navegación en base a una relación estructural entre dos clases de una *Population AIU*. Cuando se aplica este patrón, en el contexto destino se muestra la información que se relaciona con las instancias del contexto origen, a través de dicha relación estructural.

Especificación: La especificación de este patrón es similar a la del *Object Navigation AIP* salvo que, en este caso, no se utiliza un atributo de enlace. Como restricción específica, el *Relationship Navigation AIP* solo se define entre una *Manager Class* y una *Complementary Class*. Opcionalmente, puede definirse un *alias* de la relación de navegación que es mostrado al usuario para iniciar la navegación.

Semántica: A diferencia del *Object Navigation AIP*, en este patrón no se resalta ningún atributo para iniciar la navegación, sino que se añade para cada instancia un enlace que representa la navegación hacia la información relacionada con dicha instancia. Este enlace identifica la relación a través del *alias*. El uso del *Relationship Navigation AIP* implica que, la *Complementary Class* definida en la relación tiene que ser la *Manager Class,* en la *Population AIU* del Contexto de Interacción destino. Cuando se inicia la navegación, se envía al contexto destino tanto el *oid* de la instancia seleccionada como el identificador de la relación estructural. En una *Population AIU* del contexto destino, se muestran aquellas instancias de la *Manager Class* relacionadas, a través de la relación estructural que inicia la navegación, con el *oid* de instancia recibido. A diferencia del *Object Navigation AIP*, mediante el cual se recupera una única instancia de la *Manager Class*, en este caso pueden mostrar-

se un número indeterminado de instancias en función de la cardinalidad de la relación estructural.

Notación: Se utiliza la misma notación que con el *Object Navigation AIP* salvo que se omite el atributo de enlace y, en su lugar, se define opcionalmente un *alias* para la navegación.

Ejemplo: Utilizando un ejemplo similar al descrito en el patrón anterior, en este caso a partir de un investigador se desea obtener las redes de investigación a las cuales pertenece. Para definir esta interacción, se especifica una navegación relacional tal y como muestra la Fig. 4.9. De este modo, se le proporciona al usuario junto con cada ítem de información, un enlace con el nombre *subscribed Networks* que inicia la navegación. A diferencia del ejemplo anterior, en vez de mostrar la información detallada de una única red, se muestra el detalle de todas aquellas redes que estén relacionadas con el investigador.

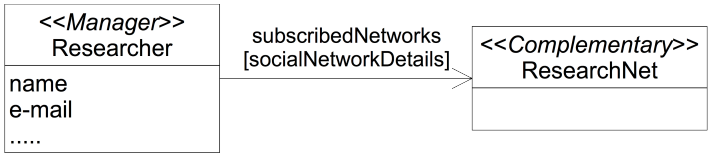


Fig. 4.9: Ejemplo de un *Relationship Navigation AIP*

4.3.3.11 Navegación de Servicio (Service Navigation)

Fundamento: Cuando en una aplicación finaliza la ejecución de un servicio o se produce un error, el usuario es redirigido al mismo contexto desde donde inició la interacción. El *Service Navigation AIP* define una navegación que se activa, automáticamente, después de la ejecución (correcta o errónea) del servicio siempre y cuando, se cumpla una condición. Al igual que los dos AIP anteriores, esta navegación lleva asociada la instancia sobre la cual se ejecutó el servicio.

Especificación: Este patrón se asocia con una *Service AIU* después de cuya ejecución, se activa la navegación hacia un Contexto de Interacción destino. El patrón especifica, además, una condición que tiene que cumplirse para que se produzca esta interacción. La condición se define en base a: (1) si el

servicio se ejecutó correctamente (*success*), (2) si se produjo algún error (*failure*) o (3) el valor de alguno de los argumentos de salida. Opcionalmente, puede asociarse una *Population AIU* perteneciente al contexto destino, de tal manera que la navegación implique al mismo tiempo una navegación por objeto.

Semántica: La diferencia de este AIP con respecto al resto de AIP de navegación es que dicha navegación se activa sin la intervención del usuario. Cuando el servicio finaliza se evalúa la condición y, si esta se cumple, se realiza la navegación. En caso contrario, el usuario se mantiene en el Contexto de Interacción donde inició el servicio. Dependiendo de si la navegación está asociada a una *Population AIU* o no, se distinguen dos comportamientos: (1) una navegación simple hacia el Contexto de Interacción o (2) un comportamiento similar a la navegación por objeto, recuperando la información relacionada con la instancia sobre la cual se ejecutó el servicio. Gracias a este último comportamiento, es posible mostrar directamente al usuario los cambios en el estado de la instancia iniciadora del servicio. Cabe destacar que la navegación por objeto no está permitida si la ejecución es errónea, porque no existe la certeza que la información de la instancia sea correcta. Por ejemplo, que se haya creado la instancia correctamente en el sistema.

Notación: el *Service Navigation AIP* se especifica, en una nueva fila de la plantilla de especificación para una *Service AIU* (Ver Tabla 4.3) denominada *Service Links.* Se definen tres parámetros: (1) una condición con la sintaxis "*if success | failure | argumentValue logicOp value*", (2) el identificador de un contexto destino y (3) opcionalmente, el identificador de una *Population AIU* del contexto destino.

Ejemplo: En la *Service AIU* definida en la Tabla 4.4, se introduce un *Service Navigation AIP*, de tal modo que cuando se añada un nuevo artículo en el sistema, se navegue al Contexto de Interacción "*ArticleDetails*". En este contexto, el usuario puede comprobar la información introducida mediante la *Population AIU* denominada "*FullArticleDetail*". Esta navegación se define tal y como se muestra en la siguiente tabla:

Tabla 4.10: Ejemplo de un *Service Navigation AIP*

Service navigations	condition	targetIC	population AIU
	if success	ArticleDetails	FullArticleDetail

4.3.3.12 Filtro Navegacional (Navigation Filter)

Fundamento: Una interacción habitual es la recuperación de los ítems de información más recientes. Desde el punto de vista de modelado, es redundante tener que definir distintos contextos, cuando únicamente varían en una restricción simple sobre la información, en este caso, la fecha de creación. El *Navigation Filter AIP* añade la semántica del *Filter AIP* a alguno de los AIP de navegación con el objetivo de definir navegaciones, que restringen de manera automática la información mostrada al usuario. En consecuencia, se filtra la información cuando el usuario accede a un contexto mediante la navegación.

Especificación: El *Navigation Filter AIP* se específica mediante una asociación con algún AIP de navegación (*Object, Relationship* o *Service Navigation*) y un *Filter AIP*. El filtro se especifica de la misma forma que el patrón correspondiente, pero como restricción adicional: tiene que definirse en función de los atributos pertenecientes a una *Population AIU* del contexto destino. Esta restricción es debida a que es en dicha AIU, donde se aplica el filtrado de manera efectiva.

Semántica: El *Navigation Filter AIP* extiende el efecto que tienen los AIP de navegación en el Contexto de Interacción destino. Mediante este patrón, se activa un filtro sin la interacción directa del usuario y que permanece activo, mientras se mantenga la interacción con el contexto. Adicionalmente, cualquier otro AIP que se utilice en el contexto, se añade al filtrado definido en la navegación. Puesto que mediante este AIP no se espera la intervención del usuario, únicamente pueden definirse fórmulas de filtrado estáticas (ver semántica del *Filter AIP*). El resto de la semántica es idéntica.

Notación: la notación para definir *Navigation Filter AIP* es una anotación, que contiene la fórmula de filtrado estereotipada con la palabra <<*filter*>>. Dicha fórmula se asocia a la asociación de navegación correspondiente

en el caso de la navegación por objeto y relacional mientras que, en el caso de la navegación de servicio, se incluye en el apartado *Service Navigations*.

Ejemplo: en nuestro ejemplo ilustrativo se ha definido un Contexto de Interacción (*MyPublications*) que muestra, a través de una *Population AIU* todos los artículos de un investigador. Sin embargo, puesto que un investigador puede tener una amplia bibliografía, es interesante disponer de un contexto que muestre únicamente los más recientes, por ejemplo los publicados durante el año en curso. Mediante el *Navigation Filter AIP*, se define esta interacción sin tener que definir nueva expresividad. La Fig. 4.10 modela este ejemplo definiendo, en primer lugar, una navegación relacional que muestra todos los artículos relacionados con el investigador. A continuación, se asocia a dicha navegación un *Navigation Filter AIP* definido sobre el atributo *publishYear* de la clase *Paper*. Activando esta navegación, solo se muestran en el contexto *MyPublications* aquellos artículos cuyo *publishYear* sea igual a *currentYear*.

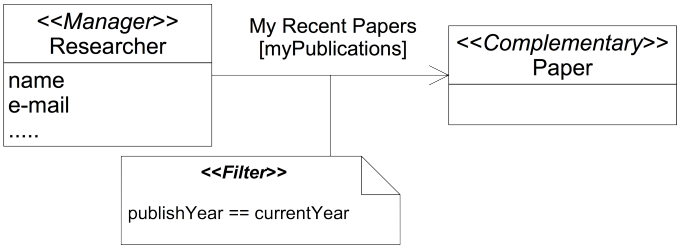


Fig. 4.10: Ejemplo de un *Navigation Filter AIP*

4.3.3.13 Edición de Instancia (Instance Edition)

Fundamento: De entre las interacciones que realiza el usuario con la información, destacan tres operaciones genéricas que se realizan frecuentemente: la creación, la modificación y la eliminación de instancias. Por esta razón, resulta interesante que el usuario pueda realizarlas de forma sencilla e intuitiva. El *Instance Edition AIP* define, simultáneamente, estas tres interacciones básicas para mantener la información que expresan las instancias de una clase.

Especificación: El *Instance Edition AIP* se asocia a la Clase Directora de una *Population AIU* sobre la cual se realizan las tres operaciones básicas descritas. Adicionalmente, el patrón se relaciona con tres *Service AIU,* para especificar una operación de creación, de modificación y de eliminación respectivamente. Esta relación no es obligatoria dado que el analista puede decidir que alguna de estas operaciones no sea soportada. Existe además otra restricción. En el caso de las operaciones de creación y modificación, para cada argumento, de las respectivas *Service AIU*, que no disponga de valor por defecto o no acepte nulos, tiene que definirse una regla de asociación. Esta regla de asociación establece, o bien que atributo de la *Population AIU* es utilizado como valor de entrada para el argumento, o bien un valor fijo que siempre es utilizado para dicho argumento.

Semántica: utilizando el modelo propuesto, si se desea realizar alguna modificación básica, como por ejemplo cambiar el valor de un atributo, tienen que definirse tanto una *Population AIU* para mostrar la información como una *Service AIU*, para ejecutar la modificación en el sistema. Este patrón permite combinar la semántica de ambas interacciones para la creación, modificación y eliminación de instancias, operaciones básicas desde el punto de vista de la lógica de negocio. En función de cómo se haya especificado el patrón, se realizan una o las tres operaciones.

Cuando el usuario selecciona una instancia, se activan las operaciones que se encuentren disponibles. En el caso de la creación, la interacción se realiza mediante la adición de un nuevo elemento de información en la interfaz, para que el usuario introduzca sus valores. Si la operación es de modificación, la interacción se realiza resaltando los valores actuales de la instancia para que el usuario los modifique. Por último, si la operación es de eliminación, se le pide la confirmación al usuario para, efectivamente, eliminar la instancia. Independientemente de la operación escogida, se realiza de forma implícita la interacción representada por la *Service AIU* correspondiente. Este hecho conlleva que otros patrones asociados a la *Service AIU,* como por ejemplo el *Validation Rule AIP*, también son tenidos en cuenta.

Las operaciones de creación y modificación están ligadas a los distintos atributos de la *Population AIU* que actúan como argumentos de entrada. Por lo tanto, existe la restricción que el usuario solo puede utilizar como argu-

mentos, aquellos atributos que se encuentren disponibles en la *Population AIU*. También se comprueba que, para cada argumento que necesite un valor, la regla de asociación proporcione un valor coherente con el tipo de datos esperado. Queda al criterio del analista establecer qué atributos son los más adecuados para proporcionar los valores necesarios en cada ejecución.

Notación: la notación del *Instance Edition AIP* se define tanto de forma gráfica como textual. Para asociar las distintas operaciones permitidas, se utiliza el apartado de operaciones de la *Manager Class* (Ver Fig. 4.11). Por otra parte, las reglas de asociación para cada argumento se definen utilizando la siguiente sintaxis XText:

```
AssociationRule:
   ServiceArgument=ID '<-' (idAttribute=ID | stValue=ID);
```

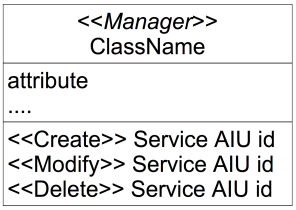


Fig. 4.11: Notación para el *Instance Edition AIP*

Ejemplo: una de las funcionalidades presentes en el ejemplo ilustrativo, es la edición colaborativa de la terminología científica. Mediante la aplicación de este AIP, se simplifica la interacción necesaria para definir este requisito. La Fig. 4.12 muestra el AIP aplicado sobre la clase directora *Terminology* cuyas instancias almacenan las distintas definiciones. Se han asociado dos *Service AIU* ("*New_Term*" y "*Update_Term*") que dan soporte tanto a la creación como a la modificación. Además, se definen un conjunto de reglas de asociación para cada servicio. Cabe señalar que los atributos *id* y *lastUpdate* son omitidos, porque es en el propio servicio donde se crean o modifican.

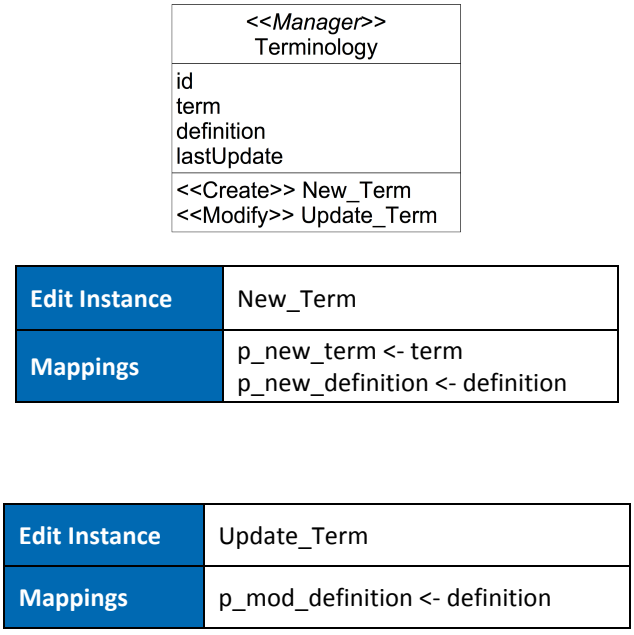


Fig. 4.12: Ejemplo de un *Instance Edition AIP*

4.4 Conclusiones

En este capítulo se ha definido un Modelo de Interacción Abstracto, para ser incluido en el método OOWS 2.0 desarrollado en el marco de la tesis. El modelado de la interacción es un tema de investigación que se ha discutido ampliamente en ámbitos académicos. En este aspecto se destaca un claro énfasis en el modelado de la IU a nivel abstracto, incluyendo algunos aspectos de interacción. Como carencia fundamental en varias propuestas, se observa la falta de un mecanismo formal para conectar dichos modelos abstractos con la funcionalidad del SI. Sin esta conexión, no es posible hablar realmente de un modelado correcto de la interacción y, lo que es más importante, es complicado generar sistemáticamente aplicaciones totalmente funcionales.

Con el fin de establecer las bases del Modelo de Interacción Abstracto, se han utilizado los modelos propuestos por los métodos OO-Method y OOWS. Dichos métodos han proporcionado la base conceptual para detectar que primitivas son útiles para el modelo a especificar. Para detectar con precisión dichas primitivas, se ha realizado un análisis comparando, desde el punto de vista de la interacción, qué semántica proporcionaba cada uno. No obstante, el Modelo de Interacción Abstracto no es solo una redefinición basada en ambos modelos, sino que también extiende ambas semánticas ofreciendo un mayor nivel de expresividad conceptual. Además, es una propuesta integradora de modo que el nuevo modelo puede ser incluido en el proceso de modelado de ambos métodos.

También se ha descrito, en el presente capítulo, el Modelo de Interacción Abstracto, detallando todas las primitivas conceptuales que lo componen, cuál es la notación propuesta para crear los modelos y cuál es la semántica esperada. Para finalizar, a modo de resumen, la siguiente tabla indica para cada primitiva conceptual introducida en el Modelo de Interacción Abstracto, que primitiva de modelado OOWS / OO-Method se ha tomado como referencia si bien, en varios casos, su semántica ha sido modificada.

Tabla 4.11: Primitivas seleccionadas para el Modelo de Interacción Abstracto

Primitiva del Modelo de Interacción Abstracto	Primitiva utilizada
User Diagram	Diagrama de Usuarios de OOWS
Interaction Map	Mapa de Navegación de OOWS
Interaction Context	Contexto de Navegación de OOWS
User Class / View	Sin equivalente
Population AIU	*Abstract Information Units* de OOWS
Service AIU	Unidad de interacción de Servicio de OO-Method
Filter	Patrón auxiliar Filtro de OO-Method
Index	Mecanismo de acceso índice de OOWS
Summary View	Vista de resultados de índices /filtros de OOWS
Order Criteria	Patrón auxiliar Criterio de Ordenación de OO-Method
Pagination	Mecanismo de Presentación Paginación de OOWS
Defined List	Patrón auxiliar Selección Enumerada de OO-Method
Validation Rule	Patrón auxiliar de Introducción de OO-Method
Argument Setting	Reglas de Dependencia de OO-Method
Object Navigation	Relación de Contexto de OOWS
Relationship Navigation	Navegación por Relación de OOWS
Service Navigation	Patrón auxiliar Filtrado Condicional de OO-Method
Navigation Filter	Patrón auxiliar Filtrado Navegacional de OO-Method
Instance Edition	Patrón auxiliar Conjunto de visualización editable de OO-Method

Capítulo 5

Modelado de IU mediante Tecnologías RIA

Este capítulo presenta una aproximación de modelado conceptual para la especificación de interfaces de usuario para una *Rich Internet Applications* (a partir de ahora Interfaz RIA). El soporte de este paradigma tecnológico por parte del método OOWS 2.0, es un factor clave para alcanzar el objetivo de desarrollar aplicaciones Web 2.0. Para tratar esta problemática, en la sección 5.1 se define, de forma precisa, que es una *Rich Internet Application* (RIA) y se justifica su estrecha relación con las aplicaciones Web 2.0. La sección 5.2 presenta un metamodelo para la creación de modelos que representen IU creadas mediante tecnologías RIA. Con dicho objetivo, se realiza un análisis para detectar los conceptos y entidades que definen una Interfaz RIA. A continuación, en la sección 5.3, se presenta una estrategia de integración del metamodelo propuesto con un método de Ingeniería Web. La sección 5.4 presenta, brevemente, un proceso generación de código para obtener una Interfaz RIA en el marco del método OOWS 2.0. Por último, la sección 5.5 presenta las conclusiones.

5.1 ¿Qué son las Rich Internet Applications?

Uno de los aspectos fundamentales del éxito de la Web 2.0 ha sido la evolución tecnológica que se ha producido en los últimos años. Esta evolución ha sido especialmente visible en las interfaces de usuario. No cabe duda, que parte la esencia de la Web 2.0 es la participación del usuario final. No obstante, sin una interfaz que resulte agradable a la par que sencilla, dicha participación no habría alcanzado el éxito que hoy contemplamos. Si analizamos las aplicaciones Web 2.0 más populares, es fácil observar que sus interfaces son mucho más complejas que las elaboradas mediante el lenguaje estándar en la Web: HTML. Este hecho es debido a las propias carencias del lenguaje HTML, que se traducen en dos problemas principales desde el punto de vista de la interacción:

- En primer lugar, el lenguaje HTML fue definido para la descripción de documentos y no para la elaboración de interfaces. Con el auge en los últimos años de las aplicaciones Web, se ha llevado la expresividad de este lenguaje al límite, utilizándolo simultáneamente para describir la información y para capturar la interacción. Además, los problemas de compatibilidad entre distintos navegadores provocan serias dificultades, a la hora de definir interfaces Web que sean realmente compatibles.

- Otro inconveniente destacable, está relacionado con las tecnologías utilizadas para generar y procesar las distintas páginas HTML. La gran mayoría de las tecnologías de desarrollo Web como ASP, PHP o J2EE se basan en el paradigma de petición-respuesta. Este paradigma implica que cualquier acción que realice el usuario en el navegador, tiene que ser comunicada al servidor, a fin de procesar una nueva pagina HTML con los cambios y retornarla al navegador. Obviamente, desde que el usuario inicia la petición hasta que ésta es completamente procesada en el servidor, existe un determinado retardo. Cuando el servidor se encuentra saturado, este paradigma provoca que la interfaz no responda de manera fluida a la interacción con el usuario.

Como solución a ambos problemas se ha impuesto el paradigma tecnológico denominado *Rich Internet Applications* o RIA. Este concepto fue en primer lugar introducido por diversos desarrolladores como Macromedia [Mullet 2003], enfatizando la necesidad que las aplicaciones Web tenían que ser más elaboradas, desde el punto de vista de la usabilidad. Aunque en un principio no se proponía una solución tecnológica concreta, se destacaban aspectos que tenían que considerarse en el desarrollo Web tales como una interacción fluida y de repuesta inmediata, el soporte multiplataforma, la conexión permanente a Internet o un diseño centrado en el usuario. En el reporte presentado por [Duhl 2003], se encuentra ya una definición más concreta de RIA: "una aplicación en la cual, todo el procesamiento de la interacción se realiza en una plataforma tecnológica cliente que se conecta, de forma asíncrona, con la funcionalidad disponible en la Web". El cambio fundamental es que el servidor ya no se encarga de procesar la IU porque ésta reside en el dispositivo cliente, por ejemplo, en un navegador Web. De este modo, es posible responder a la interacción del usuario de forma inmediata, puesto que se elimina el retardo del paradigma petición-respuesta. Además, la funcionalidad e información se proporciona de forma asíncrona, es decir, sin bloquear la aplicación hasta que se obtiene respuesta.

Este sencillo cambio de paradigma ha permitido, de manera indirecta, una evolución significativa en la forma de desarrollar interfaces en la Web. Básicamente se destaca la aparición de un conjunto de tecnologías para el desarrollo RIA, (a partir de ahora tecnologías RIA) que permiten una mayor riqueza expresiva que el lenguaje HTML. En este aspecto, se distinguen dos claras vertientes tecnológicas: (1) *Frameworks* basados en *Javascript,* que utilizan este lenguaje de programación estándar para crear componentes avanzados que guíen el desarrollo de la interfaz; (2) Tecnologías RIA específicas, que se basan en un *plug-in* o máquina virtual, instalado en el dispositivo cliente, sobre el cual se ejecuta la interfaz.

Javascript ha sido la elección habitual para desarrollar validaciones o funcionalidad sencilla en los navegadores Web, sin necesidad de acceder a la funcionalidad del servidor. Con la llegada del paradigma RIA, este lenguaje ha sido utilizado de un modo más exhaustivo hasta el punto que, hoy en día, es posible crear aplicaciones completas mediante este lenguaje. Muestra de este hecho son *frameworks* de desarrollo avanzados como *Google Web Toolkit*

[Google 2010], jQuery [Chaffer & Swedberg 2009] o EXT-JS [Sencha 2010]. Simultáneamente, estos *frameworks* ha popularizado el uso de la técnica de desarrollo Web AJAX (*Asynchronous Javascript And XML*). La función de AJAX es la de realizar peticiones bajo demanda, de la información requerida en cada momento de la interacción. Estas peticiones son realizadas en función de los eventos que desencadena el usuario, y de forma asíncrona. Las respuestas a estas peticiones son descritas mediante el lenguaje estándar XML y, procesadas utilizando *Javascript* por la parte cliente. La combinación de ambos conceptos, *Javascript frameworks* y AJAX, conforma una plataforma muy expresiva y extendida para el desarrollo de RIA.

La ventaja fundamental de esta aproximación tecnológica es el soporte casi universal del lenguaje *Javascript* en los navegadores Web. En consecuencia, se obtiene de forma casi inmediata el soporte multiplataforma. Otra ventaja es la sencilla integración entre *Javascript* y HTML, ya que los *scripts* pueden acceder a los distintos elementos de la página Web y modificarlos según sea necesario. Además, este tipo de funcionalidad puede introducirse, de forma gradual y poco intrusiva, en las aplicaciones Web actuales. Si únicamente necesitamos una funcionalidad RIA concreta, como por ejemplo comprobar si el *login* introducido ya existe en el sistema, solo es necesario utilizar el componente o *script* de *Javascript* que resuelve dicho requisito.

No obstante, el uso intensivo de *Javascript* para el desarrollo de RIA no está exento de problemas. En primer lugar, se produce el llamado efecto del "código spaghetti", puesto que el código de *Javascript* se encuentra definido en las propias páginas HTML. Este hecho produce que código de presentación y funcionalidad se encuentre entremezclado, dificultando su comprensión y mantenimiento. Asimismo, al tratarse de un lenguaje de *script* es difícil de mantener y de depurar. Este inconveniente, es una consecuencia directa de su concepción inicial debido a que fue elaborado para la definición de funcionalidad sencilla, y no como una plataforma de desarrollo. Con el auge de las RIA, se han comenzado a proporcionar entornos de desarrollo y de depuración que intentan paliar este problema. Sin embargo, estos entornos todavía distan mucho de la facilidad, que ofrecen otros entornos de desarrollo Web más asentados. Un inconveniente menor, está relacionado con el rendimiento, puesto que el código *Javascript* es interpretado por el motor del propio navegador Web. Por lo tanto, su eficiencia depende en gran medida

de la implementación del navegador utilizada. No obstante, las últimas versiones de los navegadores más populares están mejorando, continuamente, en este aspecto.

Otra aproximación para el desarrollo de RIA son las tecnologías específicas para este tipo de paradigma. Ejemplos de este tipo de tecnologías de desarrollo RIA son *Adobe Flex* [Tapper, Labriola *et al.* 2008], *Microsoft Silverlight* [Michail 2009] o *JavaFX* [Clarke, Connors *et al.* 2009]. Estas tecnologías definen, por sí mismas, un entorno completo de desarrollo RIA que incluye lenguajes de programación específicos, *frameworks* avanzados de IU, componentes para la integración con la funcionalidad y herramientas visuales de desarrollo. Por lo general estas tecnologías se centran en el desarrollo de IU avanzadas y su posterior integración con funcionalidad obtenida de forma externa. Para distribuir las aplicaciones desarrolladas no se realiza un proceso de compilación tradicional, sino que son empaquetadas y ejecutadas sobre un *plug-in* o máquina virtual, que es previamente instalada en el navegador o en el dispositivo cliente. De este modo, se evitan los problemas de los desarrollos multiplataforma, puesto que los mismos creadores de la tecnología tienen el control sobre el entorno, en el cual se va a ejecutar efectivamente la aplicación.

La principal ventaja que proporcionan este tipo de tecnologías es la riqueza, tanto visual como expresiva, a la hora de definir IU. Al ejecutarse sobre un entorno específico, es posible utilizar una amplia gama de componentes de interfaz. Como resultado, las aplicaciones RIA desarrolladas igualan a sus equivalentes de escritorio en la calidad y usabilidad de la interacción proporcionada. Actualmente, se está empezando a utilizar este tipo de tecnologías para el desarrollo de aplicaciones cliente en entornos móviles o de escritorio. Otra mejora reseñable, sobre todo con respecto a los *frameworks* basados en *Javascript*, es que estas tecnologías se encuentran perfectamente integradas en entornos de desarrollo avanzados. Así pues, se dispone de herramientas avanzadas para la codificación, depuración y mantenimiento de las RIA desarrolladas.

El hecho de ejecutarse independientemente del navegador también tiene sus inconvenientes. Habitualmente, este tipo de aplicaciones RIA son descargadas y ejecutadas automáticamente. En consecuencia, la primera vez que

el usuario accede a la aplicación tiene que esperar para su utilización, hasta que finalice el proceso de descarga. Si la aplicación tiene un gran tamaño, esta descarga puede llevar un tiempo excesivo. Otro inconveniente que cabe señalar, es que rompen la metáfora de navegación en la Web. Cuando el usuario utiliza una aplicación Web, espera una interacción predefinida en la manera de navegar a través de hiperenlaces, similar a la de las páginas Web tradicionales. Estas aplicaciones pueden modificar completamente esta interacción, como por ejemplo deshabilitar el botón para volver a la página anterior, hasta tal punto de ser completamente independientes del navegador Web. Como consecuencia, el navegador se convierte en un simple canal de distribución de la RIA.

Con la aparición de esta clase de tecnologías RIA, comienza a ser difusa la línea que separa una aplicación Web de una aplicación de escritorio. Un ejemplo de esta afirmación es la aplicación *eBay Desktop* [Ebay 2010] desarrollada mediante *Adobe Flex*. A pesar de tratarse claramente de una aplicación de escritorio, puesto que debe ser descargada e instalada en el sistema operativo, proporciona la misma funcionalidad, información e interacción que su equivalente Web. Además, ambas versiones de la aplicación comparten la misma funcionalidad (el sistema de subastas) con lo cual, salvo el entorno en el cual se ejecutan (en el navegador o en el sistema operativo), las diferencias son escasas.

Independientemente de la aproximación tecnológica escogida, las RIA han añadido un nuevo elemento de complejidad, al ya de por sí complicado ámbito del desarrollo Web. En primer lugar, han introducido la necesidad de soportar un mayor número de tecnologías en el proceso de desarrollo. Esta heterogeneidad tecnológica implica la composición de un equipo de desarrollo multidisciplinar, que domine diversas tecnologías. En segundo lugar, han puesto de manifiesto que un requisito fundamental es que la interacción mejore la experiencia del usuario. El impacto visual y la fluidez de la interacción son aspectos que en el ámbito de las RIA, marcan la diferencia entre el éxito o el fracaso.

Este nuevo tipo de aplicaciones ha originado cuestiones de investigación relevantes. Una de la cuestiones, que más interés ha despertado en los ámbitos académicos es cómo soportar el desarrollo de RIA mediante un método

de Ingeniería Web. En el campo de la Ingeniería Web [Murugesan, Deshpande *et al.* 2001], se han propuesto durante los últimos años varios métodos dirigidos por modelos para soportar el desarrollo Web. Estos métodos, varios de los cuales son descritos en [Rossi, Pastor *et al.* 2008] y reseñados en el capítulo 3 de la presente tesis, han definido una serie de modelos para soportar el desarrollo de interfaces Web. Pero tal y como ponen de manifiesto [Preciado, Linaje *et al.* 2005], se detectan una serie de carencias cuando dichos métodos son trasladados al ámbito del desarrollo RIA. En una línea similar, [Wright & Dietrich 2008] describen un conjunto de requisitos que debe soportar una aplicación RIA muchos de los cuales, no se encuentran contemplados en los métodos de Ingeniería Web actuales.

El desarrollo RIA ha propuesto mejoras en aspectos tales como la sincronización de la información cliente-servidor, el soporte multimedia o la personalización en base a las preferencias del usuario. No obstante, en el ámbito de las aplicaciones Web 2.0, es el soporte al desarrollo de IU avanzadas donde han adquirido mayor relevancia este tipo de tecnologías, por las razones anteriormente expuestas. En particular, en este capítulo se abordan los siguientes aspectos relacionados con el desarrollo de interfaces RIA:

- El modelado de una IU compuesta de componentes gráficos avanzados, o *widgets,* proporcionados por una tecnología RIA específica. Este aspecto engloba los requisitos definidos en la sección *User Interface* por [Wright & Dietrich 2008].

- El soporte a los eventos originados por el usuario a nivel de IU y, la reacción de la misma. Este aspecto se relaciona con los requisitos *Interaction* y *Synchronization* definidos por [Preciado, Linaje *et al.* 2005].

Como ya se ha mencionado, las RIA han originado otras necesidades interesantes desde el punto de vista de la Ingeniería Web, como son la sincronización de datos entre la parte cliente-servidor o, la división en distintos niveles arquitectónicos de la lógica de la aplicación [Bozzon, Comai *et al.* 2006]. Pero dichas necesidades tiene ser soportadas no a nivel de modelado, sino en los niveles de generación de código e implementación. Es en estos niveles, donde generalmente se decide que mecanismos de sincronización

son los más adecuados o, el nivel arquitectónico donde reside cada funcionalidad con el objetivo de maximizar el rendimiento del sistema.

El desarrollo de software dirigido por modelos ha ofrecido interesantes resultados, en el ámbito de la Ingeniería Web, como destacan [Koch, Meliá *et al.* 2008]. Mediante el uso de modelos conceptuales, el analista se abstrae de la complejidad subyacente de la implementación de una aplicación Web. De esta manera, puede centrarse en la definición del problema a resolver. En el ámbito del desarrollo de RIA, las ventajas que pueden ofrecer el uso de modelos conceptuales son mayores si cabe. Por ejemplo, los modelos conceptuales sirven para simplificar la complejidad tecnológica descrita en la presente sección. También tiene que considerarse que las RIA tienen una clara vertiente tecnológica que evoluciona continuamente. Si bien una implementación basada en una tecnología RIA puede quedar rápidamente obsoleta, un modelo, que describe los aspectos de interacción sin estar ligado a una tecnología, es reutilizable, más fácil de adaptar y más robusto frente a esta evolución.

Otra ventaja de introducir el modelado conceptual en el marco del desarrollo RIA, es la reutilización de los modelos introducidos en el ámbito de la Ingeniería Web. Aparte de mejorar sustancialmente la interacción con el usuario, el proceso o metodología de desarrollo de una RIA no se diferencia demasiado respecto al de una aplicación Web tradicional. Si bien las IU son radicalmente distintas, los aspectos relacionados con la funcionalidad, tratamiento de la seguridad o la personalización son similares. Si obviamos este nuevo aspecto tecnológico a nivel de interfaz, los modelos previos propuestos por la Ingeniería Web son perfectamente válidos para soportar el resto de facetas de la RIA. En consecuencia, la necesidad principal es proporcionar modelos conceptuales que representen los aspectos de interfaz relacionados con las RIA. La definición y formalización de estos modelos conceptuales, es el primer paso para integrar este nuevo paradigma tecnológico en el ámbito de los métodos de Ingeniería Web. Considerando que diversas aplicaciones Web 2.0 han sido desarrolladas y validadas utilizando tecnologías RIA, proporcionar métodos, modelos y herramientas para soportar este nuevo paradigma se convierte en una línea de investigación relevante.

5.2 Definición de un metamodelo para el desarrollo de interfaces RIA

En esta sección se define un metamodelo para la definición de IU en el marco de las RIA. El metamodelo es un artefacto software que formaliza los conceptos necesarios, para la construcción de modelos de Interfaz RIA. Por consiguiente, este metamodelo tiene que capturar toda la expresividad necesaria para definir una IU en este ámbito. A la hora de construir este metamodelo, en primer lugar, deben considerarse aquellos aspectos y entidades que definen una Interfaz RIA. A partir de esta premisa y el análisis de las interfaces de varias aplicaciones Web 2.0, se distinguen dos facetas principales a soportar por el metamodelo:

- **Componentes de IU avanzados:** la diferencia principal, entre una IU definida mediante el lenguaje HTML con respecto una definida con una tecnología RIA, es el conjunto de componentes gráficos o *widgets* que están a disposición del desarrollador. El lenguaje HTML en su versión 4 [Raggett, Hors *et al.* 1999], que es la habitualmente utilizada en el desarrollo Web, está restringido a un conjunto muy básico de componentes (*anchor, form, button, etc.*) para definir una IU. Esto es debido a que este lenguaje fue diseñado para la descripción de documentos, pero no para la definición de IU. Como consecuencia de la evolución de las IU en la Web, el lenguaje HTML ha quedado obsoleto. Una prueba de esta afirmación es el trabajo que se está desarrollando en la nueva especificación 5 de HTML [Hickson & Hyatt 2010], en fase de borrador en el momento de escribir estas líneas, que busca incluir muchas de las características que ya poseen las tecnologías RIA. La mayoría de tecnologías RIA han definido un lenguaje de descripción de IU propio, porque no se encuentran restringidas por el proceso de estandarización necesario para extender HTML. Estos lenguajes de descripción utilizan un conjunto de *widgets,* de diversa índole, que permiten mediante su composición, la definición de interfaces más avanzadas. Muchos de estos *widgets* no son originales de estas tecnologías puesto que provienen de IU definidas en entornos de escritorio. No obstante, también se han incluido *widgets* específicos para las aplicaciones Web 2.0. Ejemplos de esta clase de *widgets* son

visualizadores de contenido multimedia o editores avanzados para la creación dinámica de contenido. En consecuencia, nuestro metamodelo tiene que soportar esta variedad de *widgets*.

- **Interacción dirigida por eventos de IU**: las tecnologías RIA permiten la definición de IU altamente interactivas, es decir, que responden de manera fluida y son reactivas a las acciones que realiza el usuario sobre ellas. Esta interacción es denominada dirigida por eventos de IU, porque es iniciada mediante eventos originados por el usuario al interactuar con los distintos *widgets* que conforman la IU. Las respuestas a dichos eventos son diversas, y varían desde un cambio en la interfaz percibida, hasta la petición de nueva información desde la lógica de negocio. Este paradigma de interacción dirigido por eventos no es exclusivo de las RIA porque se ha aplicado ampliamente en aplicaciones de escritorio. A pesar de ello, no ha sido adoptado en las aplicaciones Web debido al restringido número de eventos que actualmente soporta el lenguaje HTML. Las tecnologías RIA no solo han incrementado el número de eventos relacionados con la interacción, sino que además han introducido lenguajes específicos para implementar respuestas complejas a estos eventos.

Un metamodelo que dé soporte al desarrollo de una Interfaz RIA tiene, por tanto, que dar soporte a ambas facetas. La especificación de este metamodelo se detalla a continuación.

5.2.1 Modelado de los componentes de la Interfaz RIA

Una aproximación, ampliamente aceptada en los ámbitos del modelado conceptual, es la de definir la IU como una composición de *widgets* tal y como proponen [Melia, Gomez *et al.* 2008] o [Vanderdonckt, Limbourg *et al.* 2004] entre otros. Un *widget* se define como un componente visual de la IU que proporciona los datos y gestiona la interacción con el usuario. En otras palabras, un *widget* cumple una función interactiva específica en el ámbito de la aplicación. El concepto de *widget* es recurrente en todas las tecnologías RIA, anteriormente comentadas, con lo cual se considera una primitiva conceptual esencial en el metamodelo a definir. Tomando en consideración un

conjunto amplio de aplicaciones Web 2.0 que usan tecnologías RIA, hemos clasificado los diferentes *widgets* en cinco grupos, no excluyentes, en base a la función interactiva que desempeñan:

1. **Visualización de datos** (*Dataview Widgets)*: son utilizados para mostrar al usuario un conjunto de información estructurada que es recuperada desde el sistema. Existe una gran variedad de *widgets* que cumplen con este propósito siendo las tablas y las rejillas de datos (*Datagrids*) aquéllos más utilizados puesto que muestran la información de manera concisa y clara. En el dominio de las RIA, se han incorporado además *widgets* capaces de mostrar información de carácter multimedia, gráficos avanzados así como de realizar manipulaciones de datos complejas.

2. **Introducción** (*Input Widgets*): son los *widgets* utilizados para obtener la información del usuario. En este tipo de *widgets*, existe una fuerte dependencia entre el tipo de información a introducir y el tipo de datos. De esta forma, aunque diversos *widgets* son válidos para introducir un tipo de información en concreto, desde el punto de vista de la usabilidad existen diferencias notables entre el uso de uno u otro. Un ejemplo habitual es la introducción de fechas que, si bien puede realizarse mediante un campo de texto, resulta más práctico realizarla mediante un *widget* del tipo calendario. En el ámbito de las interfaces RIA, la gran mayoría de *widgets* de introducción provienen de las aplicaciones de escritorio. Destaca, sin embargo, el denominado *Rich Text Editor* (Ver *Input Widget* en Fig. 5.1) muy habitual en aplicaciones Web 2.0 como *wikis* y *blogs*, que proporciona al usuario un editor similar a un procesador de textos para crear nuevo contenido.

3. **Navegación** (*Navigation Widgets*): este conjunto de *widgets* son utilizados para cambiar el punto de la IU que percibe el usuario en un momento determinado. La navegación resulta fundamental en el marco de la Web, ya que el enlace entre los contenidos es la base de la misma. El enlace subrayado ha sido el *widget* utilizado por excelencia para este tipo de interacción. En el ámbito de las interfaces RIA, se han introducido nuevas posibilidades como los menús contextuales o la navegación por pestañas. Además las tecnologías RIA

permiten que la navegación pueda ser potencialmente asociada a cualquier evento que el usuario realice sobre un *widget*. Sin embargo, esta flexibilidad también es contraproducente puesto que distorsiona la metáfora de la navegación y puede llegar a confundir al usuario.

4. **Servicio** (*Service Widgets)*: la misión de este tipo *widgets* es la de iniciar la ejecución de un servicio perteneciente a la lógica de negocio. Tanto en las aplicaciones de escritorio como en la Web, el botón es el *widget* más utilizado para este propósito. En el caso de la Web, es la opción más utilizada puesto que en el lenguaje HTML 4.0, su utilización lleva asociada el envío -mediante el método http POST- de información al servidor. Al igual que ocurre con la navegación, las tecnologías RIA permiten asociar a cualquier evento originado sobre un *widget* la ejecución de un servicio. A pesar de ello, para este tipo de interacción son más habituales los enlaces de botón o los iconos (Ver *Service Widget* en Fig. 5.1).

5. **Disposición** (*Layout Widgets*): a diferencia de los anteriores, la misión de estos *widgets* no es la de capturar una interacción en concreto con el usuario. Los *widgets* de disposición son utilizados para ordenar y situar, físicamente en la IU, los distintos *widgets* o crear grupos de *widgets*, que guarden algún tipo de conexión en el contexto de la tarea a realizar. Creando estos grupos, se definen interacciones que afectan a todos los *widgets* simultáneamente. En las interfaces RIA se utilizan una gran variedad de *widgets* de disposición para ocultar o resaltar algunos aspectos de la IU. Ejemplos claros son los paneles o "cajas" desplegables, para agrupar distintos *widgets* que comparten un mismo alineamiento.

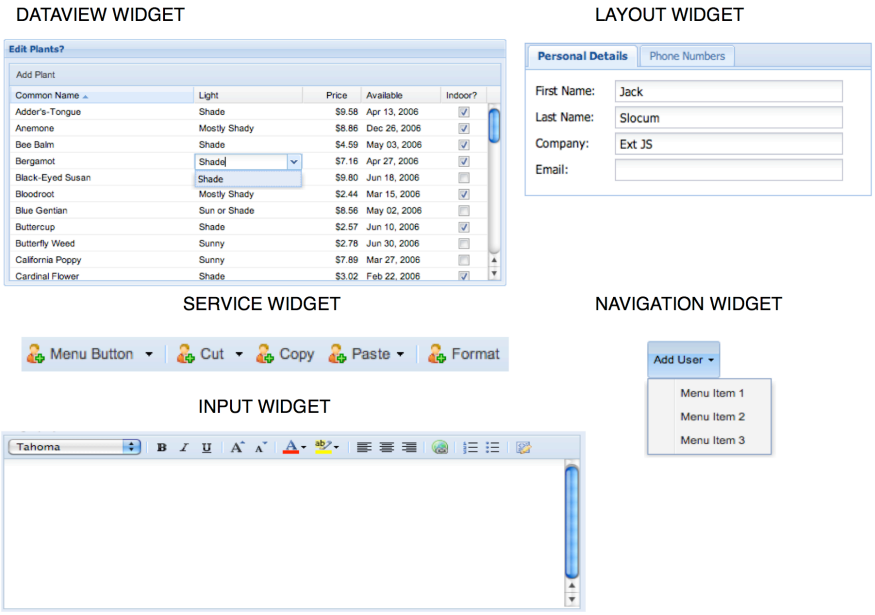


Fig. 5.1: Ejemplos de distintos *widgets* en EXT-JS

Estas cinco categorías definen la función que desempeñan los distintos *widgets* que se encuentran en una tecnología RIA. De todos modos, esta clasificación no es completamente excluyente debido a que también existen *widgets* que pertenecen a varios grupos simultáneamente. Un ejemplo muy habitual en las interfaces RIA es el *Datagrid* Editable (Ver *Dataview Widget* en Fig. 5.1). Este *widget* es una evolución del *Datagrid* básico que, además de ser utilizado para visualizar la información, se puede utilizar para la introducción, modificación y eliminación de datos. Además de la categorización de los distintos *widgets* presentes en una tecnología RIA, a partir del análisis realizado se extraen dos contribuciones, de este nuevo paradigma, desde el punto de vista de la interfaz de usuario:

- Las tecnologías RIA han aportado al desarrollo de interfaces Web una gran variedad de *widgets*, que no son completamente originales puesto que ya estaban presentes en las IU de las aplicaciones de escritorio. De esta forma, desde el punto de vista de la IU, una Interfaz RIA es una fusión de la expresividad de las interfaces Web

con las interfaces tradicionales de escritorio. Cabe destacar la existencia de algunos *widgets* cuya utilidad, principalmente, queda patente en las aplicaciones Web 2.0, como son los visualizadores multimedia o los editores de contenidos *on-line*.

- Otro aspecto relevante a destacar es que la expresividad de la Interfaz RIA está fuertemente ligada a la tecnología escogida. Por ejemplo, el tipo de IU que se puede diseñar mediante un *framework* basado en *Javascript* como *EXT-JS,* es distinta de aquella que puede conseguirse utilizando *Adobe Flex.* Esta disparidad expresiva, fruto de la constante evolución que están sufriendo las tecnologías RIA, provoca que sea complicado definir un modelo que englobe toda la expresividad que ofrece el amplio espectro de tecnologías RIA. Es cierto, que es posible la definición de este modelo en base a un conjunto de *widgets* que sea común a todas ellas. Pero con esta aproximación, se pierde la riqueza adicional que aporta cada tecnología, y que es en donde reside su principal elemento diferenciador.

Ambas ideas, al igual que la clasificación presentada, han sido tenidas en cuenta a la hora de diseñar el metamodelo propuesto. El énfasis que ponen las distintas tecnologías RIA al concepto de *widget* implica la introducción de dicho concepto a nivel de modelado. Por otra parte, si observamos que muchos de estos *widgets* provienen de las IU de escritorio, es lógico deducir que las ideas propuestas en dichos modelos son útiles también en este dominio.

Tal y como se discutió en la sección 3.1, la comunidad IPO ha abordado el modelado de la IU desde un nivel abstracto o independiente de la plataforma y, un nivel concreto o dependiente de la plataforma. Si analizamos los distintos métodos de Ingeniería Web y, en particular, el Modelo de Interacción Abstracto propuesto en la presente tesis doctoral, se concluye que desde el punto de vista de la comunidad IPO se han utilizado principalmente modelos abstractos. Es aquí donde surge el primer problema. Aunque el concepto de *widget* es en sí abstracto, guarda una estrecha relación tanto con el diseño de la interfaz como con la tecnología final empleada. Por consiguiente, carece de sentido extender los modelos actuales de Ingeniería Web me-

diante dicho concepto porque: (1) implica mezclar conceptos que pertenecen a niveles de abstracción distintos y, (2) los modelos actuales definen, de un modo preciso, los conceptos para los cuales fueron definidos y, extenderlos conlleva añadir complejidad al proceso de modelado.

Una aproximación coherente consiste en mantener dichos modelos abstractos, de navegación y presentación de la información, pero introduciendo un nuevo nivel que permita modelar la Interfaz RIA. Siguiendo la separación por niveles propuesta por la comunidad IPO, el objetivo es el de definir un nivel concreto en el cual se introduzcan estos conceptos. Incluyendo este nuevo nivel y estableciendo las correspondencias adecuadas con el nivel abstracto, es posible tratar los nuevos aspectos de Interfaz RIA pero sin mezclar distintos aspectos del modelado conceptual.

Respecto al segundo requerimiento detectado en el análisis, el soporte a la variedad ofrecida por las tecnologías RIA, la solución más directa consiste en definir un metamodelo específico, para cada tecnología que se desee soportar en el proceso de desarrollo. De esta forma, las características específicas de cada tecnología son contempladas a nivel de modelado conceptual. No obstante, no basta con crear metamodelos tecnológicos arbitrariamente, puesto que después tienen que ser incorporados y relacionados con los correspondientes metamodelos abstractos. Además muchos de los conceptos presentados, como por ejemplo la categorización de *widgets,* son independientes de la tecnología y son compartidos por todos los metamodelos tecnológicos.

En la presente tesis se propone un metamodelo para el modelado de interfaces RIA (Metamodelo RIA), que sirve de base para la especificación de metamodelos concretos para una tecnología RIA (ver Fig. 5.2). En este punto se abordan las entidades relacionadas con el diseño de la Interfaz RIA. El elemento principal de esta parte del metamodelo es el *widget* y, sus distintas especializaciones que responden a la categorización presentada. Este metamodelo RIA se compone, en primer lugar, de un conjunto de entidades abstractas, es decir, que no pueden ser directamente instanciadas para la creación de modelos, que se muestran en la Fig. 5.2 con color blanco. Una vista ampliada de estas entidades se muestra en la Fig. A.12. Este conjunto de entidades abstractas describen los conceptos que son comunes a todos los me-

tamodelos, independientemente de la tecnología RIA que representen. Dichos metamodelos tecnológicos son especificados mediante relaciones de especialización sobre estas entidades abstractas, tal y como muestra la parte inferior de la Fig. 5.2.

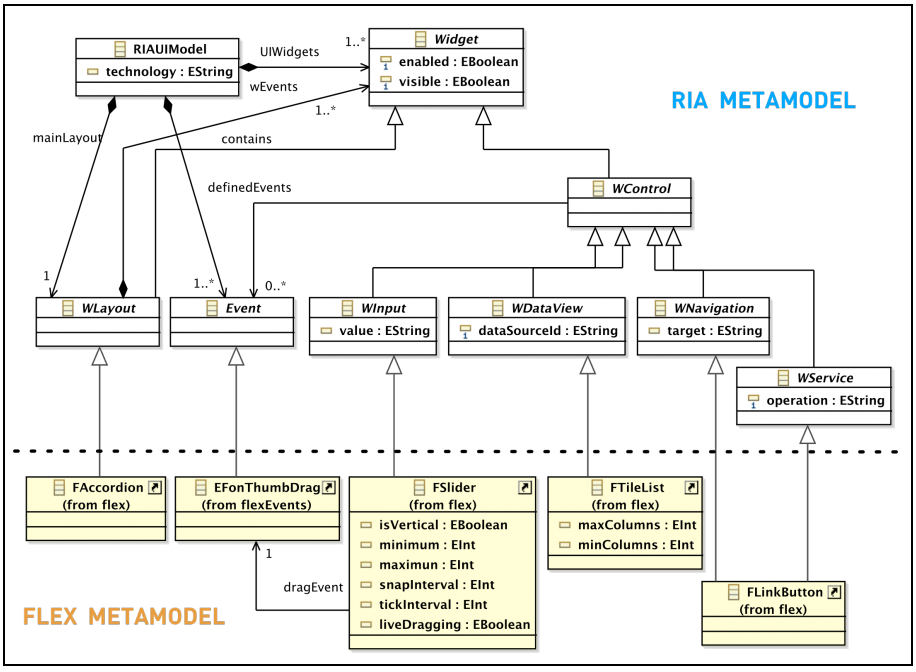


Fig. 5.2: Vista de Interfaz del Metamodelo RIA

Esta aproximación se asemeja a como se han desarrollado tradicionalmente los *frameworks* para la implementación de IU. Las entidades abstractas del metamodelo se convierten en las entidades padre, de cualquier concepto definido en un metamodelo tecnológico específico. Como consecuencia, cualquier metamodelo tecnológico comparte una serie de conceptos comunes y se encuentra articulado en base a las mismas entidades padre. Siguiendo esta aproximación, se le proporciona al analista la capacidad de definir modelos que capturen toda la riqueza de una tecnología, pero en base a unos conceptos genéricos que son comunes a todas las tecnologías RIA. Tal y como muestra la Fig. 5.2, esta aproximación se ha utilizado para la definición de un metamodelo específico de la tecnología RIA *Adobe Flex*.

Como se discute en la sección 5.3.1, esta aproximación también simplifica la conexión con los metamodelos abstractos del método de Ingeniería Web. Para establecer el enlace entre los niveles abstracto y concreto, se utilizan este conjunto de entidades abstractas que, por definición, van a estar presentes en cualquier metamodelo tecnológico. Siguiendo este razonamiento, las relaciones tecnológicas establecidas en el ámbito de un método concreto son reutilizables entre los distintos metamodelos tecnológicos. De esta manera, no es necesario crear de nuevo dichas relaciones cada vez que una tecnología RIA es incorporada al proceso de desarrollo.

A continuación se describe, en detalle, cada una de las entidades que forman parte del metamodelo RIA representado en la Fig. 5.2. Los atributos identificadores, como por ejemplo el nombre de las diferentes entidades, se han omitido para mejorar la legibilidad pero se consideran implícitos. La entidad raíz sobre la cual se define el metamodelo RIA es *RIA UI Model* que representa mediante la relación *UIWidgets*, una agregación de entidades *Widget* que son las encargadas de representar la IU. Una entidad *Widget* se define mediante un conjunto de propiedades, que son representadas en el metamodelo mediante atributos en las distintas entidades. Por ejemplo, dos propiedades que son comunes a cualquier *widget* independientemente de la tecnología utilizada, son su visibilidad, (atributo *visible* del metamodelo) y su activación (atributo *enabled* del metamodelo). Cuando la entidad *Widget* es especializada, se incluyen aquellas propiedades que sean exclusivas de una tecnología en concreto.

Las interacciones que los usuarios realizan sobre cada uno de los *widgets* son expresadas en el metamodelo mediante la entidad *Event.* Al igual que las propiedades, los distintos *Event* son especializados para soportar el conjunto específico de eventos proporcionados por una tecnología RIA. Esta entidad se relaciona con la entidad *RIA UI Model*, para su reutilización entre las distintas entidades *Widget,* ya que habitualmente, un evento es compartido simultáneamente por varios *widgets.* Esta entidad es, además, la base para definir la interacción dirigida por eventos que es presentada en la próxima sección. En el metamodelo los *widgets* son clasificados, en función de la recepción o no de eventos, mediante las entidades *WLayout* y *WControl*. La primera entidad hace referencia a los *Layout Widgets,* descritos en la clasificación, los cuales no pueden ser utilizados para la definición de la interacción basada

en eventos. Aunque es cierto que diversas tecnologías RIA si que aceptan esta posibilidad, la misión habitual de estos *widgets* es la de actuar de contenedores, por consiguiente, gracias a esta restricción se simplifica el metamodelo. En consecuencia, la entidad *WControl* hace referencia a los *widgets* que sí son susceptibles de recibir eventos.

La entidad *WControl* se especializa en otras cuatro entidades que junto a la entidad *WLayout* antes introducida, representan los cinco posibles tipos de *widgets* detectados en el análisis. *WLayout* define una entidad que sirve para agrupar, mediante la relación *contains*, al resto de *widgets* en los distintos contenedores que son mostrados al usuario. Todo *RIA UI Model* debe tener un *widget* que se encargue de contener el resto de la IU, como por ejemplo una página principal en una aplicación Web 2.0. Este es el propósito de la relación *mainLayout* que asocia, mediante una relación de cardinalidad única, el *widget* contenedor que cumple este objetivo.

Las entidades *WInput, WDataview, WNavigation* y *WService* representan, respectivamente, cada uno de los otros cuatro tipo de *widgets*. Estas entidades añaden un conjunto de propiedades para hacer referencia a los datos presentes en la interacción con el usuario. La entidad *WInput* proporciona la propiedad *value* para referenciar el valor que el usuario ha introducido. La entidad *WDataview* referencia, mediante el atributo *dataSource*, el conjunto de datos que son mostrados al usuario. La entidad *WNavigation* utiliza el atributo *target*, con el fin de establecer el destino de la navegación que representa el *widget*. Y por último, la entidad *WService* posee el atributo *operation*, que asocia la funcionalidad de la lógica negocio que activa el *widget.*

A partir del conjunto de entidades abstractas aquí descrito, se derivan los diferentes metamodelos específicos para una tecnología RIA. En la presente tesis doctoral se ha seleccionado la tecnología RIA *Adobe Flex* [Tapper, Labriola *et al.* 2008], con el objetivo de ilustrar la especificación de un metamodelo tecnológico RIA. Esta tecnología se basa en un *framework* compuesto de varios componentes que son utilizados para definir interfaces RIA complejas, y un lenguaje de programación, *ActionScript* [Lott, Schall *et al.* 2006], que permite definir lógica compleja a nivel de IU. *Flex* es una tecnología RIA pura, en el sentido que no ofrece de forma directa la capacidad de implementar lógica de negocio y persistencia. Se basa, fundamentalmente, en

proporcionar herramientas para implementar el diseño de la IU y, la integración con la funcionalidad desarrollada en otras tecnologías o proveniente en forma de servicios Web.

El *framework* de IU de *Flex* se define en base a dos tipos de conceptos: (1) *Interface Components*, que son aquéllos componentes que interactúan con el usuario y (2) *Containers,* que se encargan de definir la disposición (*layout*) de la IU. Como se observa, existe una clara relación entre los conceptos que articulan el *framework* de *Flex* y el metamodelo propuesto. Para construir el metamodelo tecnológico de *Flex,* los distintos *containers* se han definido como especializaciones de la entidad *WLayout*, mientras que los distintos *Interface Components* se han definido utilizando una entidad hija de *WControl*, según la función interactiva que desempeñan.

La parte inferior de la Fig. 5.2 muestra un conjunto de estas relaciones de especialización que definen el metamodelo tecnológico para *Flex.* Por ejemplo, el *container Accordion,* que muestra un conjunto de paneles que son mostrados u ocultados según desee el usuario, es definido como una entidad hija (en color amarillo) de *WLayout.* Por otro lado, el componente *Tilelist,* que muestra un conjunto de ítems de información en forma de mosaico, hereda de la entidad *WDataview.* En este caso, la entidad incorpora dos atributos adicionales, *maxColumns* y *minColumns*, que determinan el tamaño del mosaico. No todos los *widgets* tecnológicos se definen mediante una única relación de especialización. Por ejemplo, el componente *linkButton* puede desempeñar tanto la función de navegación como la función de ejecutar un servicio. Por esta razón, este componente hereda simultáneamente de las entidades *WNavigation* y *WService*. Al igual que los *widgets* y las propiedades, los distintos eventos tienen que ser creados mediante relaciones de especialización. En el ejemplo, el componente de tipo *WInput* denominado *Hslider*, se encuentra relacionado con el evento específico de *Flex* denominado *thumbDrag*. Este componente representa una barra de desplazamiento, para la selección de un valor incremental según un intervalo definido por la propiedad *snapInterval.* Cuando el usuario mueve dicha barra de desplazamiento, se produce el evento *thumbDrag*. El metamodelo tecnológico completo puede consultarse en el Anexo A.3.

Cabe señalar que el objetivo del metamodelo tecnológico es el de proporcionar un mecanismo de modelado, para configurar determinados parámetros de la generación de la IU. De esta manera es factible adaptarla, desde una perspectiva tecnológica, a los requisitos de interacción de la aplicación. Por ejemplo, para la introducción de un valor, existen multitud de *Input widgets* disponibles. Dependiendo de la información a introducir, por parte del usuario, existen *widgets* más adecuados que otros desde el punto de vista de la usabilidad. La elección por parte del analista del *widget* tecnológico más adecuado es modelada en este nivel de abstracción.

El hecho de trabajar con modelos conceptuales solventa en parte el problema de introducir esta expresividad en un método de Ingeniería Web. En primer lugar tiene que considerarse cual es el verdadero objetivo de este metamodelo: concretizar las primitivas conceptuales de los niveles más abstractos del método en una representación coherente de la tecnología RIA seleccionada. Este objetivo se aborda con mayor detalle en el punto 5.3, pero básicamente la idea es establecer una relación entre una primitiva conceptual del método y la IU de usuario que la representa. Por esta razón no es necesario soportar los *widgets* sin una correspondencia con alguna primitiva conceptual de los modelos del método.

Tampoco es el objetivo de este metamodelo representar, exhaustivamente, todos los atributos o eventos que soportan un conjunto de *widgets* tecnológicos. Por ejemplo, muchos de los *widgets* que proporciona *Flex* poseen propiedades que permiten variar su apariencia visual tales como el color, el tamaño o la tipografía. Aunque las propiedades estéticas influyen en la percepción que tiene el usuario de la interfaz, la interacción no cambia. Es decir, si el usuario tiene que introducir un valor, la interacción es la misma sea cual sea el tamaño del *widget* o la fuente tipográfica utilizada. Por consiguiente, únicamente tienen que soportarse aquellas propiedades que configuran o restringen la interacción. Proporcionando un metamodelo tecnológico simplificado, es más sencilla la labor del analista puesto que se abstrae, en mayor medida, la complejidad de la tecnología RIA subyacente.

Respecto a la notación para representar este modelo, existen ciertas restricciones que hacen inviable una propuesta concreta. Al tratarse de un modelo de carácter tecnológico es necesario definir una notación, obviamente,

ligada a la tecnología. Este hecho provoca que para cada modelo derivado a partir del metamodelo, tenga que definirse una nueva notación. La construcción de un editor visual para cada notación es una tarea que resulta costosa. Por otro lado, desde el punto de vista de la especificación, también resulta poco eficiente que el analista tenga que introducir todos y cada uno de los *widgets*, que conforman la IU.

La aproximación propuesta en la presente tesis doctoral para evitar esta necesidad, se basa en la generación automática de un modelo conceptual genérico de la IU, a partir de cada una de las primitivas conceptuales abstractas que se hayan definido. En otras palabras, a la vez que se construyen los respectivos modelos abstractos que especifican la interacción, se genera un modelo tecnológico a modo de versión preliminar. Al encontrarnos en un entorno dirigido por modelos es factible, a partir de este modelo preliminar, la generación automática de una Interfaz RIA para su visualización. Por dicha razón, es la propia IU final la notación que se propone para nuestro modelo tecnológico. La ventaja es evidente porque esta aproximación ofrece una representación visual fiel de la IU final al analista. Además, si se seleccionan primitivas conceptuales por defecto adecuadas, no es necesario que el analista defina todo el modelo de IU desde cero. En consecuencia, la labor del analista se reduce a la modificación de dicha IU preliminar, a fin que se adapte a las necesidades específicas de la interacción a representar. Este proceso se detalla, en el marco del método OOWS 2.0, en el punto 5.3.2.

5.2.2 Modelado de la interacción dirigida por eventos de IU

En la anterior sección, se ha especificado las entidades del metamodelo RIA para el diseño de IU basadas en componentes provenientes de tecnologías RIA. No obstante, una IU implementada en una tecnología RIA no es solo una mera composición estática de *widgets*. Además del diseño de la interfaz, también tienen que abordarse las reacciones a la interacción del usuario y, cómo son modificados algunos aspectos de la interfaz percibida. Debido a que los eventos producidos sobre los distintos *widgets* son los iniciadores de estos cambios, este tipo de interacción ha sido denominada como dirigida por eventos de IU. Para definir un metamodelo para el desarrollo de interfaces RIA, la expresividad para soportar este tipo de interacción tiene que ser

incluida. Con dicho fin, las distintas reacciones posibles a los eventos producidos han sido especificadas a nivel de modelado. Para catalogar estas reacciones, se han analizado un conjunto de aplicaciones Web 2.0:

- **Cambios en la IU**: la reacción más habitual ante la ocurrencia de un evento es que se produzca un cambio en la IU percibida por el usuario. Estos cambios se manifiestan sobre las distintas propiedades que definen un *widget*, de tal forma que varia tanto su representación visual como su función interactiva. Por ejemplo, ante un evento, es posible activar o desactivar la interacción que proporciona un *widget*.

- **Petición de datos bajo demanda**: esta reacción es muy común en las aplicaciones RIA, sobre todo en aquéllas que hacen un uso intensivo de AJAX. La reacción consiste en pedir a la lógica de negocio, un conjunto de datos específicos como consecuencia de un evento producido por el usuario. Un ejemplo típico de esta reacción es el patrón de "autocompletado". Este patrón consiste en recuperar, únicamente, los valores que cumplen con una cadena de texto introducida por el usuario. Conforme el usuario pulsa el teclado, se genera un evento que inicia esta reacción pidiendo los datos.

- **Ejecución de funcionalidad**: esta reacción implica una comunicación de petición-respuesta con la lógica de negocio. A diferencia de la reacción anterior, en este caso la reacción no se limita a la recuperación de información. Además, se trata de una comunicación síncrona, es decir, la interacción con la aplicación se detiene hasta que se recibe la respuesta. Es habitual mostrar un mensaje de progreso mientras la ejecución se produce y, un mensaje de *feedback* cuando esta finaliza.

- **Retroalimentación (*feedback*) de validación**: esta reacción informa al usuario sobre errores que se han producido como consecuencia de la interacción, proporcionando una posible solución. Son varios los eventos de las aplicaciones Web 2.0 que se asocian a esta reacción, desde la introducción de valores erróneos, por ejemplo que se haya introducido una fecha sin el formato adecuado, a validaciones complejas que tienen en cuenta la lógica de negocio, por ejemplo, comprobar si el identificador de usuario introducido no ha sido utilizado previamente.

- **Navegación**: esta reacción implica un cambio completo de la IU percibida por el usuario. A diferencia de la primera reacción descrita, que afecta únicamente a un conjunto de propiedades de la IU percibida, una navegación cambia el punto de la aplicación en el cual se encuentra el usuario y, en consecuencia, el conjunto de interacciones disponibles. Cabe reseñar, que en las tecnologías RIA se han añadido distintos tipos de efectos visuales para enfatizar este cambio al usuario.

A diferencia de la clasificación realizada en la sección anterior, que englobaba todos los posibles *widgets* de una Interfaz RIA, en este caso, las reacciones definidas no son las únicas que se encuentran en las aplicaciones Web 2.0. Por ejemplo, además de la retroalimentación por validación, es común la retroalimentación de ayuda que tiene el objetivo de guiar al usuario cuando usa la interfaz. Por lo tanto, el número de reacciones posibles no está acotado a un conjunto predefinido. No obstante, estas cinco reacciones clasifican las más frecuentemente utilizadas y, aquéllas que se han considerado más útiles para abordar el desarrollo de una aplicación Web 2.0. Adicionalmente, es preferible definir un metamodelo más simple de utilizar que añadir reacciones poco habituales.

Con el objetivo de definir esta interacción dirigida por eventos, se introduce a nivel de metamodelado el concepto de regla de evento (*Event rule).* Una regla de evento se define en base a un evento perteneciente a un *widget* origen, el cual desencadena una reacción que afecta a la interfaz y a un *widget* destino de la reacción. Este concepto guarda una cierta similitud con las llamadas reglas ECA (*Event-Condition-Action rules*), que han sido tradicionalmente propuestas por la comunidad IPO, por ejemplo en [Mbaki, Vanderdonckt *et al.* 2008].

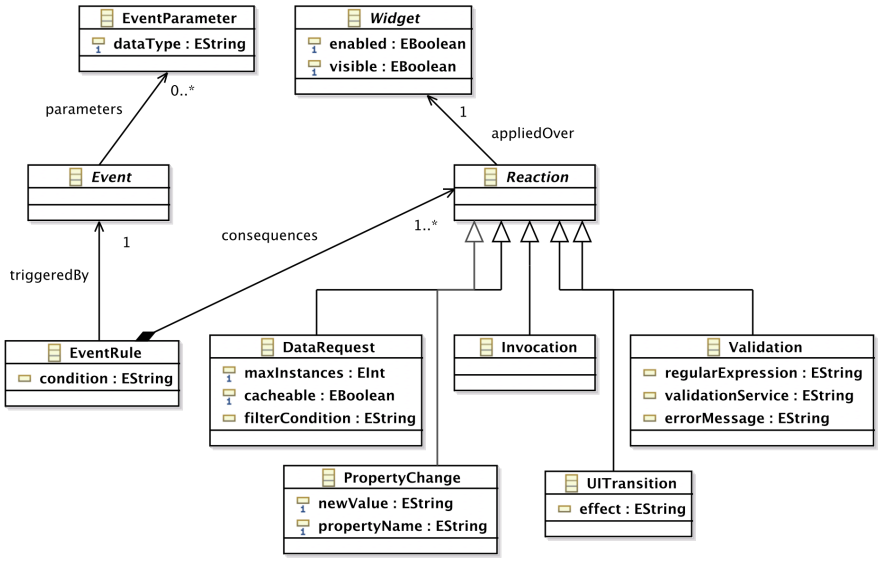


Fig. 5.3: Vista de *Event Rules* del Metamodelo RIA

La Fig. 5.3 muestra la distintas entidades que definen esta parte del metamodelo. Al igual que en la figura anterior, los identificadores de la entidades se han omitido para mejorar la legibilidad. Una regla de evento (entidad *EventRule*) se define mediante la asociación con un evento (relación *triggeredBy*) y, al menos, una o varias reacciones (relación *consequences*). Opcionalmente, puede definirse mediante el atributo *condition* una condición adicional a cumplir para que la regla se aplique. El ámbito de las reglas de evento es la interfaz sobre la cual se encuentra el *widget* cuyo evento inicia la reacción. De este modo, una regla de evento puede generar múltiples reacciones sobre *widgets* diferentes, siempre y cuando, se encuentren contenidos en el mismo punto de la IU en el cual se originó el evento. Siguiendo este razonamiento, el *widget* sobre el cual se origina el evento y aquel que recibe la reacción pueden ser el mismo. También, los eventos pueden incluir parámetros que son utilizados para configurar las distintas reacciones. Estos parámetros son definidos mediante la entidad *EventParameter*, y su tipo es especificado mediante el atributo *dataType*.

Cada reacción es representada en el metamodelo, mediante una entidad especializada de la entidad *Reaction*. Además, cabe destacar que no todas las

reacciones pueden ser asociadas a cualquier *widget* ya que, dependiendo de su función interactiva, la reacción puede no estar representada por dicho *widget*. Por ejemplo, no tiene sentido asociar una reacción para la ejecución de funcionalidad a un *widget* utilizado para definir la disposición de la IU. Las cinco entidades que describen las reacciones soportadas por el metamodelo son las siguientes:

1. ***Property Change:*** representa un cambio en un valor cualquiera de las propiedades del *widget* sobre el cual se aplica la regla. Mediante esta reacción, el estado de un *widget* es controlado en base a una sucesión de eventos. Dos atributos de la entidad controlan este cambio: (1) *propertyName*, que referencia la propiedad del *widget* destino que es modificada y (2) *newValue*, que define el nuevo valor que tiene la propiedad cuando se aplica la regla de evento. Puesto que todos los *widgets* son descritos usando propiedades, potencialmente, se puede aplicar esta reacción a cualquiera de ellos.

2. ***Data Request***: esta reacción representa una recuperación de información desde la lógica de negocio, con el objeto de mostrarla mediante un *widget* de visualización de datos. El conjunto de datos a recuperar está definido por la información que el *widget* represente. La entidad dispone de tres atributos que modifican el comportamiento de esta reacción: 1) *cacheable,* si este atributo está definido como cierto, no se recupera de nuevo aquella información que ya se encuentre disponible localmente. Este mecanismo, sin embargo, puede generar problemas de coherencia si dicha información fue modificada, 2) *maxInstances,* define el número máximo de ítems de información que son recuperados al aplicar la reacción y 3) *filter condition,* especifica una fórmula de filtrado, basada en una expresión lógica, para seleccionar qué información es recuperada.

3. ***Invocation***: esta reacción representa la ejecución de un servicio definido en la lógica de negocio de la aplicación. Por lo tanto, la reacción siempre se encuentra asociada a un *widget* que soporte la ejecución de servicios. La entidad proporciona un único atributo para especificar si la ejecución del servicio es asíncrona o síncrona, es decir, si el usuario puede interactuar o no con la IU mientras la ejecución está

en progreso. Además por defecto, esta reacción genera un evento *onServiceResult* que informa de la correcta ejecución o no del servicio. Este evento puede ser utilizado para lanzar reglas de evento en función del resultado, correcto o erróneo, de la ejecución.

4. ***Validation***: esta reacción proporciona una retroalimentación al usuario sobre, si alguno de los datos que ha introducido, contiene algún error. En consecuencia, se asocia con aquellos *widgets* utilizados para la introducción de información. Esta reacción toma alguno de los valores presentes en el evento que la ha originado, para comprobar si es coherente. La reacción se define en base a tres atributos: (1) *regular Expression,* define una expresión regular basada en la sintaxis PCRE [Hazel 2010], que tiene que ser satisfecha por el valor a comprobar, (2) *validationService,* establece una referencia con un servicio de la lógica de negocio que es utilizado para validar el valor y, (3) *errorMessage,* define una descripción textual que es proporcionada al usuario, con el fin de corregir el error.

5. ***UI Transition****:* esta entidad representa una transición en la IU que percibe el usuario. Una transición se define en base al nuevo *widget* que contiene la IU a mostrar al usuario y, un *widget* de navegación que se encarga de iniciar esta transición. A fin de enfatizar la reacción al usuario, se especifica, mediante el atributo *effect* un efecto visual, es decir, una animación, resaltar la información, etc. que informe de la transición, siempre y cuando, la tecnología RIA utilizada soporte dicho efecto.

La definición de las reglas de evento se realiza mediante una notación textual que instancia el metamodelo propuesto. Para formalizar esta notación se ha utilizado XText [Behrens, Clay *et al.* 2009], un *framework* para el desarrollo de lenguajes específicos de dominio (DSL). Utilizando este *framework,* se ha definido una sintaxis y una gramática para la definición de reglas de evento, que es equivalente al metamodelo propuesto. Una vez definida la gramática, el *framework* se encarga de generar, automáticamente, un editor textual para la definición y la validación de las reglas de evento. La ventaja principal, que proporciona el uso de XText, es que la gramática se representa como un metamodelo *Ecore* al igual que el resto del metamodelo RIA. Así

cuando el analista especifica las reglas de evento, en un segundo plano, se construye un modelo conceptual que representa dichas reglas de evento. En consecuencia, la integración entre la versión textual de la reglas de evento y la versión conceptual es factible.

La Fig. 5.4 muestra la gramática que se ha definido, en la presente tesis doctoral usando el lenguaje XText. En líneas generales, la definición de una regla de evento se compone de cinco bloques cada uno de ellos encabezado mediante una palabra reservada. El orden y la semántica de cada uno de estos bloques se describe a continuación:

- *DEFINE*: en este bloque se declaran las distintas reacciones que participan en la regla de evento indicando su tipo.

- *IF*: especifica una condición que tienen que cumplir los *widgets* sobre los cuales se define la regla para que sea ejecutada.

- *SET*: en este bloque se especifican, en primer lugar, las distintas reacciones con el *widget* sobre el cual son aplicadas y, opcionalmente, se inicializan las propiedades de dichas reacciones.

- *ON:* define el evento encargado de iniciar la ejecución de la regla de evento.

- *DO*: describe el efecto de la regla mediante una llamada a la interacción, que abstrae el *widget* sobre el cual se asocia la reacción. Este bloque contiene, opcionalmente, un conjunto de cambios en los *widgets*, en las reacciones o en otros elementos del Modelo de Interacción Abstracto como, por ejemplo, los valores de los argumentos de entrada.

```
/* Event Rule main structure */
EventRule:
    define_rule=DEFINE_side
    (condition_rule=IF_side)?
    set_rule=SET_side
    on_rule=ON_side
    do_rule=DO_side;

/* Auxiliary rules */
Reactions:
    'DataRequest' | 'Invocation' | 'Transition' |
    'Validation' | 'Change';

Reaction_expression:
    targetWidget=ID'.'reactionType=Reactions'();';

Assign_expression:
    leftSide=Property_expression ('='|'+=')+
    rigthSide=(Value_expression|ID) ';';

Property_expression:
    name=ID'.'property=ID;

terminal Value_expression:'"'(.)+'"';

/* Definition of reactions of the Event Rule */
DEFINE_side:
    'DEFINE: ' (reactionType=Reactions name=ID ';')*;

/* Optional condition to trigger the rule */
IF_side:
    '\nIF: ' checkProperty=Property_expression
    check=LogicOp (ID|Value_expression|ID'.'ID)';';

LogicOp:
    '=='|'!='|'>'|'<'|'>='|'<='|'inList'|'notInList';
```

```
/* Link definition between widgets and reactions */
SET_side:
    '\nSET: ' ((reaction=ID ' TO ' targetWidget=ID ';') |
                (initializations+=Assign_expression) )*;

/* Event that triggers the rule */
ON_side:
    '\nON: '  event=Event_definition;

Event_definition:
    sourceWidget=ID'.'event=Event_expression;

Event_expression:
    name=ID'('(parameters+=ID ",")* (parameters+=ID)');';

/* Reaction to perform */
DO_side:
    '\nDO: ' (propertiesChanges += Assign_expression)*
           reaction += Reaction_expression;
```

Fig. 5.4: Gramática para la definición de *Event Rules*

La Fig. 5.5 ilustra una regla de evento que tiene como objetivo simular el patrón *Suggestion*. Este patrón muestra al usuario un conjunto de valores textuales, que coinciden con un texto que está introduciendo, con el fin de agilizar el proceso de introducción. En esta regla, en primer lugar, se define una reacción de tipo *DataRequest*. A continuación, se establece una condición para que únicamente se active la regla, si el *widget WList,* el cual es utilizado para mostrar el conjunto de sugerencias, se encuentra activado. Mediante la instrucción *SET*, se asocia la reacción definida con el *widget WList* y, se define el atributo *cacheable* como cierto, para evitar recuperar continuamente los mismos datos.

Con el fin de iniciar la regla se utiliza el evento *onDataChange*, que pertenece al *widget* de introducción denominado *WTextInput*. Este evento se produce cada vez que el usuario cambia el contenido de dicho *widget,* de tal manera que el nuevo valor se encuentra disponible en el parámetro *newValue*. En la sentencia *DO,* se configura la propiedad *filterCondition* de la reacción, para que la fórmula de filtrado se actualice con el nuevo valor del parámetro *newValue*. Finalmente, mediante la sentencia *WList.DataRequest* ( ), se recupera el nuevo conjunto de elementos que muestra el *widget WList*.

```
DEFINE:
    DataRequest RequestReaction;
IF:
    WList.enabled == "true";
SET:
    RequestReaction TO WList
    RequestReaction.cacheable="true";
ON:
    WTextInput.onDataChange(newValue);
DO:
    RequestReaction.filterCondition="LIKE newValue";
    WList.DataRequest();
```

Fig. 5.5: Ejemplo de la definición de una *Event Rule*

A diferencia del metamodelo propuesto para el diseño de una Interfaz RIA, tanto el metamodelo como la notación propuesta para las reglas de evento es utilizada entre las distintas tecnologías RIA. La principal justificación es que todas estas tecnologías soportan el concepto de evento a nivel de IU. Además, proporcionan la expresividad necesaria para implementar el conjunto de reacciones propuesto. La única diferencia tecnológica es, por lo tanto, el conjunto de eventos que cada *widget* soporta. A pesar de ello, tanto el metamodelo como la propia gramática de las reglas de evento son lo suficientemente flexibles, para solventar esta diferencia.

5.3 Aplicación del metamodelo RIA en un método de Ingeniería Web

Los modelos actuales de los métodos de Ingeniería Web contienen información y expresividad relevante, a la hora de generar una Interfaz RIA. Por ejemplo, los modelos abstractos que proponen se encargan de especificar las distintas estructuras de datos utilizadas por el SI. Estas estructuras de datos son, después, utilizadas en la interfaz para representar la información. En consecuencia, existe una estrecha relación entre los modelos que describen la información y los modelos que representan la IU. En la sección anterior, se presentó un metamodelo para soportar tanto el diseño como la interacción dirigida por eventos de una Interfaz RIA. Sin embargo, este metamodelo no es un artefacto aislado, en el marco de un método de producción de software,

debido a que se relaciona con el resto de modelos e influye en el proceso de generación de código.

En esta sección, se propone una estrategia para introducir el metamodelo RIA propuesto en el marco de un método de Ingeniería Web dirigido por modelos. La estrategia se ha definido de forma independiente a cualquier método de Ingeniería Web. No obstante, a fin de validarla convenientemente, se ilustra su aplicación en el marco del método OOWS 2.0. Además se detalla un proceso de generación de código de dicho método, con el objetivo de obtener una IU basada en la tecnología *Adobe Flex*.

5.3.1 Estrategia de conexión a nivel de metamodelado

Como punto de partida, disponemos de un conjunto de modelos a un nivel abstracto, propuestos por un método de Ingeniería Web, y un modelo a un nivel tecnológico, definido en base al metamodelo RIA propuesto. Utilizando la terminología propuesta por el estándar MDA de la OMG [Mellor, Scott *et al.* 2002], se establece una clara correspondencia entre los conceptos de PIM (*Platform-independent model*) y PSM (*Platform-specific model*). Esta situación es habitual tanto en los métodos DSDM como en los métodos de Ingeniería Web. La solución más habitual para establecer las correspondencias entre ambos niveles, es la definición de un proceso de transformación entre modelos, tal y como proponen [Melia, Gomez *et al.* 2008] o [Koch 2006]. Este proceso se basa en un conjunto de reglas de transformación modelo a modelo (M2M), que generan a partir del modelo abstracto o PIM, una versión equivalente del modelo tecnológico o PSM. Por lo tanto, una vez finalizado el proceso de transformación, el modelo tecnológico sustituye al modelo abstracto a partir del cual ha sido generado. A partir de ese punto, únicamente el modelo tecnológico es utilizado en el resto del proceso de desarrollo.

Aunque desde el punto de vista teórico esta aproximación resulta coherente, en la práctica supone dos problemas a resolver:

- En primer lugar, para que el proceso de transformación sea válido, los modelos tecnológicos tienen que soportar toda la expresividad de los modelos abstractos. Solo así, se garantiza que los modelos

tecnológicos son capaces de representar la misma expresividad que los definidos en el nivel abstracto. Si se tiene en cuenta este hecho, la definición de estos modelos tecnológicos resulta compleja debido a que tienen que combinar la expresividad de ambos niveles.

- En segundo lugar, una primitiva conceptual del nivel abstracto no tiene una única relación con una primitiva del nivel más tecnológico. Por ejemplo, una primitiva conceptual del nivel abstracto puede ser representada en la IU final mediante distintos *widgets*, siempre y cuando, soporten la interacción abstraída. Como consecuencia directa, no es posible definir un conjunto acotado de transformaciones M2M, que se apliquen de forma sistemática en todos los escenarios.

Con el objetivo de evitar esta problemática, la solución propuesta en la presente tesis doctoral es la omisión de este proceso de transformación M2M. En su lugar, se propone la definición de relaciones entre los metamodelos que representan ambos niveles de abstracción. De esta manera, para cada primitiva conceptual del nivel abstracto, se especifica que primitiva conceptual del nivel tecnológico es la más adecuada para la interacción a representar. Con el objetivo de capturar estas relaciones a nivel conceptual, se propone la creación de *weaving models* tal y como se propone en el trabajo de [Fabro, Bézivin *et al.* 2006].

Un *weaving model* [Didonet Del Fabro & Valduriez 2009] es un modelo conceptual que captura las relaciones existentes entre otros dos modelos conceptuales, que no comparten el mismo metamodelo. En otras palabras, un *weaving model* relaciona dos modelos que pertenecen a metamodelos distintos. Un *weaving model* se construye en base a un *weaving metamodel*. Como consecuencia, este metamodelo referencia los dos metamodelos que especifican los modelos a relacionar. El trabajo propuesto por [Didonet Del Fabro & Valduriez 2009] presenta un *weaving metamodel* genérico, mostrado en la Fig. 5.6, que relaciona dos metamodelos cualesquiera, con el objetivo de especificar un *weaving model*.

El elemento raíz del metamodelo propuesto es *WModel,* que se compone de los distintos elementos que conforman el *weaving model,* básicamente, enlaces y referencias a las primitivas conceptuales de los otros metamodelos.

Las entidades *WLink* y *WLinkEnd* definen las relaciones entre los modelos, mediante la creación de enlaces con primitivas conceptuales. Para referenciar, de un modo preciso, qué primitivas conceptuales de cada modelo se relacionan, la entidad *WElementRef* establece el identificador de dichas primitivas, unívocamente, mediante el atributo *ref*. Dicho atributo se utiliza en una función de identificación, a fin de retornar la primitiva conceptual enlazada.

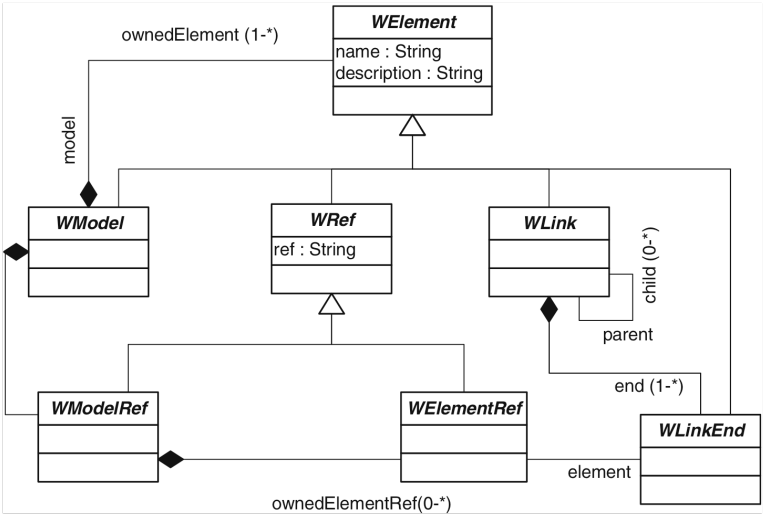


Fig. 5.6: *Core Weaving Metamodel*

La principal ventaja del uso de un *weaving metamodel* es que los dos metamodelos que son relacionados no tienen que ser modificados. Toda la información para llevar acabo la integración, se formaliza en el *weaving metamodel*, de tal forma que las correspondencias son independientes a nivel conceptual. Cuando el analista construye un *weaving model*, en cierta manera, está definiendo un proceso similar, para relacionar el nivel abstracto con el nivel concreto, al tratado mediante una transformación M2M. La diferencia fundamental es la omisión de un metamodelo que combine la expresividad de ambos niveles. Otra razón que justifica el uso de *weaving models,* es su satisfactoria aplicación en distintos ámbitos de la Ingeniería del Software, como son el diseño de líneas de producto [Cetina, Fons *et al.* 2008], o la definición de la arquitectura de sistemas [Jossic, Del Fabro *et al.* 2007].

En el marco de la presente tesis doctoral, se utilizan los *weaving models* con el fin de establecer la correspondencia entre las primitivas conceptuales del Modelo de Interacción Abstracto y, las entidades de un Modelo de Interfaz RIA. Tomando las ideas propuestas por [Didonet Del Fabro & Valduriez 2009], se ha especificado un *weaving metamodel*, especificado en el lenguaje *Ecore,* que enlaza el metamodelo de interacción abstracto presentado en la sección 4.3, con el metamodelo presentado en la sección 5.2.1. En nuestro caso particular, el *weaving metamodel* propuesto no es genérico, puesto que se definen asociaciones explícitas con el metamodelo de interacción de OOWS 2.0. La Fig. 5.7 muestra las entidades principales que forman el metamodelo especificado, para ilustrar su uso.

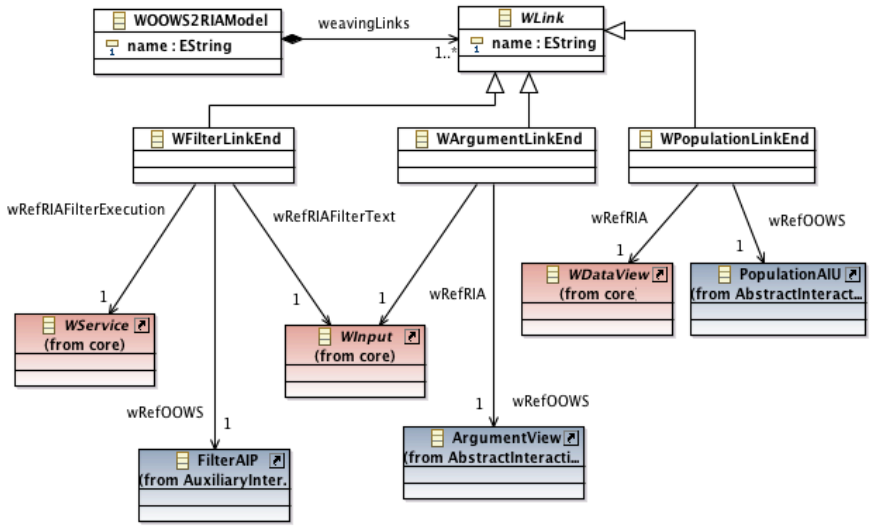


Fig. 5.7: *Weaving Metamodel* para OOWS 2.0

El *weaving metamodel OOWS2RIA* se compone de un conjunto de *weaving links* que son representados mediante la entidad abstracta *WLink*. Esta entidad se especializa en distintas entidades *WLinkEnd* que se componen de, al menos, dos asociaciones unidireccionales: (1) una primera asociación denominada *wRefOOWS,* que hace referencia a una primitiva conceptual del Modelo de Interacción Abstracto (en color azul) y (2) una o varias asociaciones denominadas *wRefRIA*, que hacen referencia a primitivas conceptuales del metamodelo de Interfaz RIA (en color rojo). En consecuencia, a diferen-

cia del metamodelo propuesto en [Didonet Del Fabro & Valduriez 2009], no se utiliza una función de identificación para establecer las referencias, sino que se definen explícitamente a través de la definición de asociaciones. Esta decisión implica que el *weaving metamodel* propuesto es específico para un método de Ingeniería Web. A pesar de ello, esta decisión facilita tanto la creación como el posterior uso de los *weaving models* en el proceso de desarrollo.

Un aspecto a destacar es que las entidades del metamodelo de Interfaz RIA, que se utilizan en el *weaving metamodel*, no son primitivas conceptuales específicas de una tecnología RIA. En particular, las relaciones se establecen con las entidades abstractas, que catalogan cada una de las funciones interactivas que un *widget* desempeña, presentadas en la sección 5.2.1. Por ejemplo, la Fig. 5.7 muestra una *Population AIU* que se relaciona con un *widget* de visualización de datos (*WDataview*), cuando se especifica el *weaving model*. Otro ejemplo mostrado es la relación con el *Filter AIP*. En este caso, la interacción que se representa implica que el usuario introduzca un valor y, posteriormente, ejecute el filtrado en sí. En consecuencia, este patrón se asocia con dos *widgets*, representados por las entidades *WInput* y *WService,* para soportar adecuadamente esta interacción. Este razonamiento es aplicado, sistemáticamente, para el resto de primitivas conceptuales pertenecientes al Modelo de Interacción Abstracto. Para resumir la construcción de este *weaving metamodel*, la Tabla 5.1 muestra el conjunto de relaciones definidas, a través de *weaving links* entre ambos metamodelos. La columna uno muestra una primitiva conceptual del Modelo Interacción Abstracto mientras que la columna dos, la entidad abstracta del Modelo de Interfaz RIA que la representa. Por último, la columna tres muestra un *widget* de la tecnología *Adobe Flex*, seleccionado por defecto, para representar la interacción.

Tabla 5.1: Relaciones de *weaving* con el Modelo de Interacción Abstracto

Primitiva del Modelo de Interacción Abstracto	Entidad/es Abstracta/s	Widget/s FLEX
Interaction Map	WNavigation	LinkBar
Interaction Context	WLayout	Panel
Population AIU (simple)	WDataView	DataGrid
Population AIU (master-detail)	WDataView	AdvancedDataGrid
AttributeView (text)	WDataView	Text
AttributeView (image)	WDataView	Image
AttributeView (video)	WDataView	VideoDisplay
Service AIU	WLayout WService	Form Button
Argument View	WInput	TextInput
Argument View (DateTime)	WInput	DateChooser
Argument View (Boolean)	WInput	CheckBox
Argument View (Text)	WInput	TextArea
Filter (static)	WService	LinkButton
Filter (dynamic)	WInput WService	TextInput LinkButton
Index	WNavigation	LinkBar
Summary View	WDataView	List
Order Criteria	WService	ToggleButtonBar
Pagination	WNavigation	LinkBar
Defined List	WInput	ComboBox
Validation Rule	WDataView	Alert
Object Navigation	WNavigation	LinkButton
Relationship Navigation	WNavigation	LinkButton
Instance Edition	WDataView	AdvancedDataGrid

La integración mediante *weaving models* tiene otra clara ventaja asociada. Al definir los distintos metamodelos tecnológicos RIA, mediante especializaciones con estas entidades abstractas que clasifican los *widgets*, el *weaving metamodel* definido para un método específico puede ser reutilizado entre distintas tecnologías RIA. La razón es que el *weaving metamodel* relaciona cada primitiva conceptual con una función interactiva, pero no con un *widget*

tecnológico en concreto. Un *WLinkEnd* se asocia siempre con una de estas entidades abstractas. De esta manera, cuando se crea el *weaving model* se tiene que instanciar dicha relación. La creación de dicha relación solo es posible utilizando una entidad hija especializada a partir de dicha entidad abstracta, ya que por definición, las entidades abstractas no pueden utilizarse directamente para la creación de un modelo. En consecuencia, instanciando los *WLinkEnds* definidos, se selecciona que *widget* tecnológico se relaciona a una primitiva del Modelo de Interacción Abstracto.

Tal y como muestra la Fig. 5.8, para las entidades *SummaryViewAIP* y *InteractionContext* se establece, respectivamente, una relación de *weaving* con las entidades *WDataView* y *WLayout*. Si se selecciona como tecnología RIA a utilizar *Adobe Flex,* estas relaciones implican que, por ejemplo, el patrón *Summary View* se representa en la IU final, o bien mediante un *Datagrid*, o bien con un *Treeview*. Por otra parte, los Contextos de Interacción son representados mediante cualquier contenedor, como por ejemplo *Box* o *Accordion*. Si se utiliza otra tecnología RIA, la relación de *weaving* se mantiene salvo que, en este caso, el conjunto de *widgets* que pueden ser seleccionados es distinto.

Qué *widget* tecnológico es el más adecuado tiene que ser decidido por el analista, a la hora de crear las correspondientes relaciones de *weaving*. La tarea del *weaving metamodel* es la de restringir que *widgets* son factibles en base a su función interactiva, pero no determina si el *widget* es válido en el contexto que se desea utilizar. Utilizando el ejemplo anterior, el patrón *Summary View* puede ser representado mediante el *widget VideoDisplay,* cuya función es la de mostrar información multimedia, debido a que este *widget* también cumple con la función de visualizar información. A pesar de ello, la utilización de este *widget* no tiene sentido, si la información a mostrar no es de carácter multimedia. Para solventar este problema, se tienen que definir un conjunto de restricciones semánticas para validar que los *weaving models* definidos son correctos, desde el punto de vista de la generación de la IU. Estas restricciones se definen como precondiciones al proceso de generación de código y, son implementadas a través de la herramienta de soporte al método.

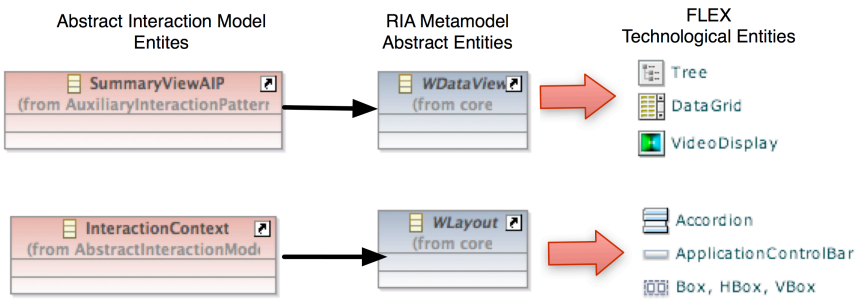


Fig. 5.8: Relación entre los distintos niveles conceptuales

La especificación de los *weaving models* se realiza utilizando el editor proporcionado por EMF, de manera similar al resto de modelos. La Fig. 5.9 muestra la definición de una relación de *weaving*, utilizando el editor de modelos generado a partir del *weaving metamodel* presentado. Previamente a la definición de dicha relación, se han definido tanto una *Population AIU* como un *widget* de tipo *DataGrid*, especificando los metamodelos correspondientes. A continuación, mediante la definición de una entidad *WPopulationLinkEnd*, se crea la relación de *weaving* instanciando, adecuadamente, las propiedades *WRefOOWS* y *WRefRIA*. Cuando se instancia la propiedad *WRefRIA*, se selecciona cualquier *widget* del Modelo de Interfaz RIA de *Flex*, que se especialice a partir de *WDataView*.

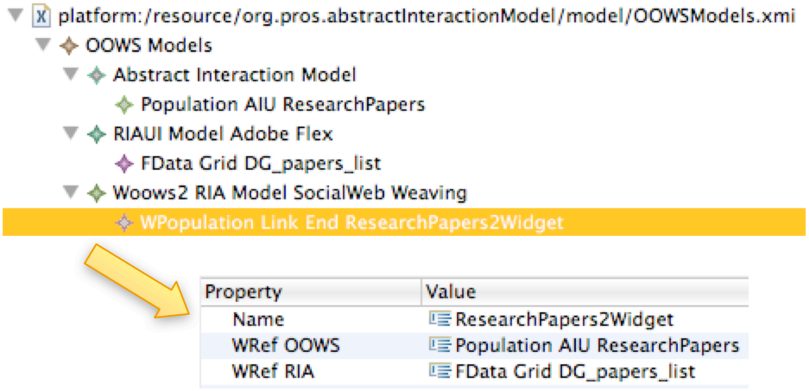


Fig. 5.9: Edición de un *weaving model* con Eclipse EMF

Respecto a la definición de los *weaving models*, deben considerarse algunos aspectos adicionales. En primer lugar, no todas las primitivas conceptuales del método tienen que asociarse a un elemento del modelo tecnológico. Por ejemplo, en el ámbito del método OOWS 2.0, la primitiva conceptual *User*, aunque es relevante desde el punto de vista de la interacción, no tiene una representación directa en la IU con lo cual, no tiene sentido que pertenezca a una relación de *weaving*. Por otra parte, primitivas conceptuales, que no son parte implícita del Modelo de Interacción Abstracto, pueden tener una correspondencia con la IU. Un ejemplo concreto, es la primitiva *Argument* que representa el argumento de un servicio proveniente del Modelo de Objetos OO-Method. Si bien esta primitiva conceptual no pertenece al Modelo de Interacción Abstracto, sino al Modelo de Objetos de OO-Method, el tipo de *Argument* influye en el *widget* utilizado para la introducción del valor. Por lo tanto, dicha primitiva tiene que estar presente a la hora de definir el *weaving model* correspondiente.

En segundo lugar, un problema asociado a esta aproximación es la dificultad a la hora de construir los *weaving models*. Este hecho se produce porque el analista tiene que definir una relación de *weaving*, para cada primitiva conceptual del modelo que es representada en la IU. Si el modelo posee un elevado número de primitivas, esta tarea resulta laboriosa. Un mecanismo para simplificar este proceso es el de establecer un proceso de generación automático de relaciones de *weaving*. En multitud de escenarios es posible, a partir únicamente del Modelo de Interacción Abstracto, generar una IU genérica que represente la interacción. Para conseguir este propósito, tienen que asumirse una serie de transformaciones por defecto, que asocian a cada primitiva conceptual, el *widget* que soporta un mayor número de escenarios. De hecho algunas de las aproximaciones analizadas en el estado del arte, utilizan esta aproximación como única propuesta para la generación de código de la interfaz. En nuestro caso, se define un proceso similar para simplificar la definición de *weavings*.

A partir de un Modelo de Interacción Abstracto, es posible definir un conjunto de transformaciones M2M que generen, automáticamente, una primera versión de la IU formada por un Modelo de Interfaz RIA y el modelo de *weaving* correspondiente. De este modo, se obtiene una primera versión de la IU sin necesidad que el analista cree, explícitamente, un *weaving*

model. A partir de este modelo inicial, el analista únicamente tiene que modificar aquellas relaciones, que no se correspondan con el *widget* que desea utilizar. La columna tres de la Tabla 5.1 muestra un conjunto de *weaving* genéricos, definidos para el método OOWS 2.0, cuando la tecnología RIA utilizada es *Adobe Flex*. La utilización de estas transformaciones genéricas tiene otra ventaja: la utilización de la información implícita en el Modelo de Interacción Abstracto, con el objetivo de determinar qué *widget* es él más adecuado. Gracias a este hecho es posible definir distintas transformaciones, que proporcionan el *Input Widget* más adecuado, en función del tipo de datos del argumento a introducir. A continuación, se explica con mayor detalle esta aproximación, enmarcándola en la fase de modelado conceptual del método OOWS 2.0.

5.3.2 Modelado conceptual de la Interfaz RIA

Una vez presentados los distintos metamodelos que conforman nuestra propuesta, se describe a continuación, la fase de modelado conceptual que se sigue para obtener la especificación una Interfaz RIA. Esta fase se resume en la Fig. 5.10 a través de su aplicación al método OOWS 2.0. El proceso se ilustra, por lo tanto, utilizando conceptos propios del método OOWS 2.0. No obstante, tanto los metamodelos como la estrategia basada en *weaving models*, son aplicables en otro método de Ingeniería Web, puesto que se han definido de forma genérica. A continuación, se detallan cada uno de los pasos que conforman esta fase de modelado conceptual.

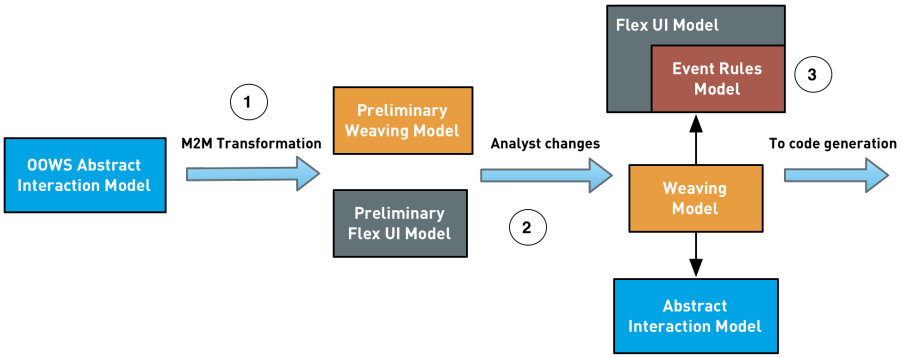


Fig. 5.10: Modelado conceptual de la Interfaz RIA en OOWS 2.0

1. Generación del modelo preliminar de Interfaz RIA

Este proceso comienza a partir del Modelo de Interacción Abstracto que define los distintos Contextos de Interacción. La construcción de este modelo se realiza mediante el editor EMF asociado al metamodelo presentado en la sección 4.3. Como paso previo a la construcción de este modelo, se define un modelo OO-Method, mediante la herramienta *OLIVANOVA*, que describe la funcionalidad del sistema de forma precisa. De los tres modelos que proporciona OO-Method, únicamente el Modelo de Objetos es utilizado en esta etapa, puesto que es el encargado de expresar la interfaz con la lógica de negocio y las estructuras de datos de la aplicación.

Una vez definido el Modelo de Interacción Abstracto, se procede a la generación de una versión preliminar de dos modelos (Paso 1 de la Fig. 5.10): (1) un Modelo de Interfaz RIA, acorde con el metamodelo propuesto en el presente capítulo y (2) un *weaving model,* que relaciona este Modelo de Interfaz RIA con el Modelo de Interacción Abstracto que se ha especificado. Esta generación se realiza automáticamente, mediante una transformación M2M, derivando a partir del Modelo de Interacción Abstracto especificado, una versión preliminar de ambos modelos. Previamente a la generación, el analista tiene que seleccionar la tecnología RIA destino, por ejemplo *Adobe Flex*.

Como paso opcional, puede seleccionarse el conjunto de transformaciones M2M a aplicar, de entre las que proporcione el proceso de desarrollo. En

consecuencia, es posible generar distintas versiones preliminares del Modelo de Interfaz RIA. Gracias a este mecanismo, el analista selecciona que transformaciones son más convenientes, en función de la interfaz a desarrollar. Como abordar el soporte de múltiples conjuntos de transformaciones queda fuera del alcance de la presente tesis doctoral. No obstante, puede consultarse el trabajo de [Aquino, Vanderdonckt *et al.* 2008] para obtener más detalles al respecto.

2. Modelado de la interfaz basada en una tecnología RIA

Como salida del paso anterior, se dispone de un modelo preliminar de la IU. Sin embargo, ya que dicho modelo se ha generado a partir de un conjunto de transformaciones genéricas, es previsible que no se ajuste completamente a las necesidades de la interacción a desarrollar. En ese caso, se utiliza como un modelo inicial para no tener que crear desde cero la especificación completa de la IU. De esta manera, se simplifica y minimiza la labor de modelado del analista. Para la adecuación de este modelo, en primer lugar, el analista tiene que crear el conjunto de *widgets* que desea utilizar para representar cada uno de los Contextos de Interacción. Estos *widgets* se crean instanciando las entidades correspondiente del metamodelo tecnológico seleccionado, en este caso el metamodelo RIA para *Adobe Flex*. Opcionalmente, el analista puede configurar algunos aspectos de los *widgets,* a través de las distintas propiedades que proporcionan sus correspondientes primitivas conceptuales.

Una vez definido el conjunto de *widgets* que representa un Contexto de Interacción, se crean las relaciones de *weaving* que enlazan las primitivas conceptuales presentes en el Modelo de Interacción Abstracto, con los *widgets* definidos en el Modelo de Interfaz RIA. Estas relaciones se crean, o bien modificando, o bien extendiendo el *weaving model* preliminar obtenido en el paso anterior. Por ejemplo, una *Population AIU* es representada por defecto mediante el *widget Datagrid*, tal y como muestra la Tabla 5.1. Para modificar esta representación, en primer lugar, se instancia un *widget* que soporte la recuperación de información. A continuación, se modifica la entidad *WPopulationLinkEnd* del *weaving model* (ver Fig. 5.7) para referenciar al nuevo *widget*. El resultado final es una versión modificada de ambos modelos (Paso 2 de la Fig. 5.10).

3. Modelado del comportamiento de la interfaz

Siguiendo la aproximación presentada, una Interfaz RIA reacciona a los eventos que el usuario genera sobre la misma. En consecuencia, después de definir qué *widgets* tecnológicos definen la interfaz, tiene que modelarse este comportamiento. Como se explica en el punto 5.2.2, este comportamiento se define mediante un conjunto de reglas de evento o *Event Rules*. A diferencia de los modelos anteriores, no es posible generar una versión preliminar de estas reglas puesto que: (1) el comportamiento de la interfaz se encuentra ligado a los *widgets* tecnológicos que el analista haya introducido, explícitamente, en el paso anterior y, (2) el Modelo de Interacción Abstracto no aporta la expresividad necesaria para derivar correctamente este aspecto de la interacción.

La definición de estas reglas se realiza utilizando el editor textual que se genera a partir de la gramática *XText* mostrada en la Fig. 5.4. Como paso previo a la definición de las reglas, se importa en este editor el Modelo de Interfaz RIA definido, a fin de acotar el conjunto de eventos disponibles. Cuando el analista define las reglas de forma textual, implícitamente, se construye un modelo que es enlazado con el Modelo de Interfaz RIA (Paso 3 de la Fig. 5.10). Este enlace es factible, dado que tanto el Modelo de Interfaz RIA como el modelo que representa las *Event Rules*, están integrados en el mismo metamodelo. El resultado final del proceso es un Modelo de Interfaz RIA extendido que incluye el comportamiento de la interfaz. Este modelo de interfaz extendido, junto al *weaving model* y el Modelo de Interacción Abstracto, es utilizado en el proceso de generación de código que se describe a continuación.

5.4 Generación del código de la Interfaz RIA: *Adobe Flex*

La última etapa del proceso de desarrollo es la generación del código final de una interfaz RIA. En la presente sección se describe, en líneas generales, cómo dicho proceso es llevado a cabo en el método OOWS 2.0. En esencia, el proceso aquí descrito es una extensión de la aproximación presentada en [Valverde 2007], pero incluyendo los modelos tecnológicos que describen la

interfaz RIA. Las entradas de este proceso de generación son: (1) un Modelo de Interacción Abstracto, (2) un modelo de interfaz, específico de una tecnología RIA, que incluye su comportamiento y, (3) un *weaving model* que relaciona ambos modelos. El proceso de generación se realiza, sistemáticamente, generando el código mediante un conjunto de reglas de transformación de modelo a código (M2C). Estas reglas son definidas mediante el lenguaje *Xpand* que pertenece al *framework OpenArchitectureWare* [OAW 2008].

Una transformación en el lenguaje *Xpand* se basa en un conjunto de reglas, asociadas a una plantilla textual de código. Cada una de estas reglas especifica, a efectos prácticos, el fragmento de código que se obtiene a partir de una primitiva conceptual del metamodelo. Por ejemplo, en nuestro caso particular se tiene que definir una regla, para cada AIU y AIP definidos en el Modelo de Interacción Abstracto. El encabezado de una regla se compone de un nombre que la identifica, un conjunto de parámetros opcionales y el nombre de la clase del metamodelo (por ejemplo, *Population AIU*) sobre la cual se aplica. *Xpand* incorpora una propiedad *this* implícita, a partir de la cual se referencian por su identificador los atributos de la metaclase y, otros elementos del metamodelo relacionados con ésta. El lenguaje también incorpora instrucciones propias de un lenguaje de programación imperativo, tales como bucles *foreach* o sentencias condicionales *if*, que ayudan a definir reglas más complejas. Además, incluye operadores avanzados para la selección, el manejo de colecciones y la navegación a través de la información del modelo. De esta forma, el cuerpo de la regla se compone de una plantilla que define la salida textual del proceso de generación, en nuestro caso código *Flex*, y otras expresiones del lenguaje.

La generación del código mediante *Xpand,* se basa en la concatenación de llamadas a reglas, que van produciendo código para cada primitiva conceptual del metamodelo. Ese proceso comienza en la regla definida como *root*. Para invocar una regla desde otra, se utiliza la sentencia *Expand*, que recibe el nombre de la regla cuya salida textual va a expandirse junto a los parámetros definidos. La sentencia *Expand* recibe también un elemento del metamodelo que debe ser del tipo que espera la regla. Este elemento se referencia mediante el operador *for*, si es único, o mediante *foreach* si es una colección de elementos obtenidos, por ejemplo mediante una relación de cardinalidad *n*, sobre los cuales tiene que aplicarse la regla. Por ejemplo, en la regla

que se define sobre el Mapa de Interacción de un usuario se utiliza el operador *foreach*, para invocar la regla de generación de código, de cada uno de los Contextos de Interacción que forman el mapa.

A continuación, se describe, brevemente, los distintos bloques de código que se generan, como resultado de la aplicación del proceso de generación de código. El resultado final de este proceso es una aplicación *Flex,* compilada como un fichero *swf,* que se comunica con la lógica de negocio generada mediante el compilador de modelos de OO-Method, encapsulada en un componente COM+.

Código visual de la IU

Este código representa la parte visual, compuesta de *widgets*, perceptible por el usuario. En *Adobe Flex,* la especificación y disposición de los distintos *widgets* se realiza mediante el lenguaje MXML. Este lenguaje específico de la plataforma *Adobe Flex,* es un lenguaje de especificación de IU declarativo, basado en el estándar XML, similar a otras aproximaciones como USIXML [Vanderdonckt, Limbourg *et al.* 2004]. En MXML, cada *widget* se representa mediante un nodo del fichero XML, de tal modo que sus propiedades son representadas mediante atributos XML. La Fig. 5.11 muestra en la parte izquierda un fragmento de código MXML y la IU asociada en la parte derecha.

Las reglas de generación de esta parte del código siguen el siguiente patrón. Para cada entidad del Modelo de Interacción Abstracto, se consulta, a través del *weaving model*, que primitiva conceptual del Modelo de Interfaz RIA es utilizada para representar la interacción. A continuación, esta primitiva conceptual es utilizada como entrada para la invocación de una regla, que representa el *widget* tecnológico mediante una plantilla de código. Esta regla se compone, fundamentalmente, de un fragmento de código MXML mediante el cual, los atributos del *widget* presentes en el metamodelo RIA son instanciados, en función de los valores utilizados en el modelo. Este proceso se repite para cada primitiva conceptual del Modelo de Interacción Abstracto construyendo, en cada iteración, un nodo del fichero MXML correspondiente. Una ventaja del paradigma declarativo de MXML es que el orden

en el cual se generan cada uno de los nodos del fichero final, no influye en la IU resultante.

```
<mx:Panel x="19" y="10" width="265" height="263" layout="absolute"
    title="CheckOut" id="panel1">
    <mx:RadioButtonGroup id="rg_tipoPago"/>
    <mx:RadioButton x="32" y="36" label="PayPal" groupName="rg_tipoPago"
        click="currentState='st_paypal'" id="radiobutton1"/>
    <mx:RadioButton x="32" y="62" label="CreditCard" groupName="rg_tipoPago"
        click="currentState='st_credito'" id="radiobutton2"/>
    <mx:LinkButton id="lk_continuar" x="170" y="201" label="Continue"
        click="currentState='st_envio'" rollOverEffect="{ef_glow}"/>
    <mx:Canvas x="15" y="92" width="220" height="101" id="cv_extendido">
    </mx:Canvas>
</mx:Panel>
```

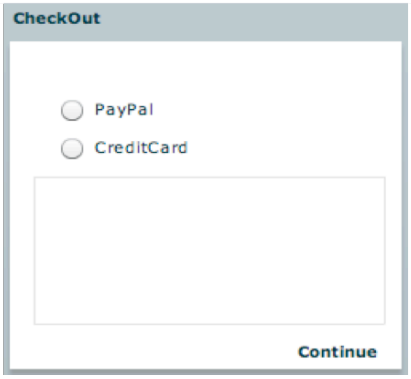


Fig. 5.11: Ejemplo de código de IU en Flex

Código de comportamiento de la IU

Este código implementa las reacciones de la IU que han sido definidas mediante las *Event Rules* correspondientes. A diferencia del código anterior, para la implementación del comportamiento se utiliza el lenguaje *ActionScript*. Este lenguaje sigue el paradigma de programación imperativo y el orientado a objetos, proporcionando una sintaxis similar a la ofrecida por *JavaScript*. Por lo tanto, es posible definir clases, objetos así como funciones, utilizando los constructores habituales de este tipo de lenguajes. En el presente proceso de generación de código, se define un conjunto de *scripts* (utilizando el nodo XML *script* del lenguaje MXML) que codifican el comportamiento de la IU.

En *Flex,* cada uno de los eventos de un *widget* tiene asociado un atributo en el nodo correspondiente del código MXML. Por ejemplo, en la Fig. 5.12, el nodo XML del *widget TextInput* tiene un atributo *change*, que referencia el evento que se produce cuando el usuario cambia el valor de entrada del *widget*. Asociado a este atributo, se define un *event listener*, el cual implementa una función en *ActionScript* a ejecutar como reacción, cuando el evento se produce. Utilizando el mismo ejemplo, se observa que el evento *change* está asociado a la función *onDateBeforeChange.* Este *event listener* simplemente codifica una reacción, del tipo *Property Change,* que actualiza al día siguiente, una fecha que es almacenada en otro *widget* presente en la IU.

```
<mx:Script>
<![CDATA[
private function onDateBeforeChange( event:Event ) : void {

 var dateAfter:date;
 dateAfter = dateBefore + 1;
 txtDateAfter.text = dateAfter.toString();
}
]]>
</mx:Script>

<mx:TextInput id="txtDateBefore" change="onDateBeforeChange(event)" />
```

Function that implements the event reactions

Property change

Link between event and the widget

Fig. 5.12: Ejemplo de código generado en *ActionScript*.

Una vez analizado cómo se implementan en *Flex* las reacciones a eventos, la implementación de las reglas de generación de código es inmediata. En este caso, las reglas de generación de código tienen como entrada las *Event Rules* asociadas a cada uno de los *widgets.* En primer lugar, por cada evento de un *widget* perteneciente a una *Event Rule,* se define un nuevo *event listener* como una nueva función privada. El identificador del *event listener,* el cual corresponde con el identificador de la *Event Rule* modelada, se utiliza como valor del atributo que representa el evento en el nodo MXML correspondiente. El código que ejecuta este *event listener* se genera en base a la reacción definida en la *Event Rule*. Para cada tipo de reacción presente en el metamodelo, se especifica una plantilla genérica de código *ActionScript* que codifica el cuerpo de cada *event listener.* Puesto que la lógica de las reacciones

no es muy compleja y el metamodelo proporcionado es lo suficientemente expresivo, la generación del código del cuerpo del *event listener* resulta factible.

Código de integración con OO-Method

En el método OOWS 2.0, la lógica de negocio modelada en OO-Method se produce mediante la herramienta comercial *OLIVANOVA*. Con el objetivo de integrar la interfaz generada con la funcionalidad, tiene que analizarse, previamente, el resultado del proceso de generación de código de *OLIVANOVA*. Aunque el código resultante varía dependiendo de la tecnología destino seleccionada (.NET o J2EE), la arquitectura de la aplicación generada se divide en dos componentes claramente diferenciados: (1) un componente cliente, que encapsula la interfaz modelada mediante el Modelo de Presentación OO-Method y (2) un componente servidor, que encapsula toda la funcionalidad modelada junto al acceso a la Base de Datos. En consecuencia, una aplicación OO-Method está formada por dos componentes software, basados en el mismo conjunto de modelos conceptuales, pero independientes desde el punto de vista de la implementación. En nuestro caso particular, se desecha el componente cliente obtenido, ya que no se utiliza el Modelo de Presentación de OO-Method para generar la interfaz, sino los modelos introducidos en la tesis doctoral.

Si bien la documentación oficial de *OLIVANOVA* enfatiza el uso de una API propietaria a efectos de integración, la comunicación entre ambos componentes (cliente-servidor) se realiza a más bajo nivel. Esta comunicación se produce mediante un proceso de intercambio de mensajes XML de petición/respuesta que consta de los siguientes pasos:

1. En primer lugar, el componente cliente construye un mensaje XML, mediante la API para realizar, por ejemplo, una consulta de los investigadores dados de alta.

2. Este mensaje se envía al componente servidor donde se procesa y se invoca a la operación demandada de la clase correspondiente (*Researcher.Query*, por ejemplo).

3. El resultado de la operación se vuelve a codificar como XML, y es enviada al componente cliente como respuesta.

4. La respuesta es recibida por el componente cliente, que se encarga de procesar la información e enviarla a las clases de la interfaz como objetos.

Para abstraer esta comunicación con *OLIVANOVA*, la solución adoptada es la de crear estos mensajes mediante una fachada de negocio implementada como un Servicio REST [Richardson & Ruby 2007]. La elección de la implementación de la fachada de negocio como un Servicio REST, se fundamenta en el amplio soporte que ofrecen las tecnologías RIA a este paradigma de comunicación. Este servicio implementado ofrece una serie de operaciones genéricas, para acceder a la funcionalidad de cada una de las clases que componen la lógica de negocio. Para cada clase del Modelo de Objetos, el servicio REST ofrece una URL, a partir de la cual se ejecutan las siguientes operaciones:

- *QueryPopulation*: devuelve la población de una clase determinada. Recibe como argumento una colección con los identificadores de los atributos a mostrar.

- *QueryById*: devuelve una única instancia de una clase a partir de su *id*. Recibe como argumentos el identificador de la clase, una colección con los identificadores de los atributos a mostrar y el identificador único de la instancia a recuperar.

- *QueryRelated*: a partir del *id* de una instancia, devuelve todas las instancias pertenecientes a otra clase relacionada. Recibe como argumentos una colección con los identificadores de los atributos a mostrar de la clase relacionada, el identificador único de la instancia de la clase origen y el identificador de la relación entre las clases origen y destino.

- *ExecuteService*: ejecuta una operación perteneciente a la lógica de negocio, a partir del identificador unívoco de un servicio y el conjunto de sus argumentos. Como argumento se envía el valor (codificado como una cadena de texto) y el tipo de datos. El método

retorna cierto, si la ejecución del servicio fue correcta, o falso en caso contrario.

- *GetServiceResponse*: recupera el resultado o el error del último servicio ejecutado en la fachada de negocio. No recibe argumentos.

La implementación de este servicio se ha realizado en PHP utilizando una parte del *framework* presentado en [Valverde 2007]. Esta fachada de negocio es reutilizable ente distintas interfaces puesto que no está ligada con la funcionalidad específica de una aplicación. Además, la integración con Servicios REST está ampliamente implementada en diversas tecnologías RIA. Por estas razones, no es necesario introducir esta integración de forma explícita en el proceso de generación de código.

La utilización por parte de la UI de este servicio se describe a continuación. Cuando una componente de la interfaz solicita una funcionalidad de la capa de lógica, se invoca, mediante REST, la operación correspondiente de la fachada de negocio. Dicha fachada se encarga de construir y enviar el mensaje XML que acepta el componente servidor. Una vez recibida la respuesta, en forma de otro mensaje XML, ésta es procesada por la misma fachada de negocio y almacenada en las estructuras de datos que espera la interfaz.

Una vez detallada como es realizada la integración con la funcionalidad, se pasa a describir el código que tiene que generarse a nivel de interfaz para invocar al Servicio REST. La aproximación seguida es la de generar una clase *ActionScript*, por cada clase del Modelo de Objetos que es utilizada en algún tipo de interacción, o bien a través de la consulta de sus atributos, o bien a través de la ejecución de una de sus operaciones. Esta clase actúa como mediadora o *wrapper* de la lógica de negocio, proporcionando a la IU una representación local de la funcionalidad. Para cada clase se incluye una operación, que permite la recuperación y actualización de los atributos, y un conjunto de operaciones, por cada servicio de la lógica de negocio utilizado en el Modelo de Interacción Abstracto. Cuando la interfaz necesita alguna funcionalidad, simplemente realiza un acceso a estas clases *wrapper*. Sin embargo, en un segundo plano, se invoca a través de las operaciones definidas en la clase *wrapper* correspondiente al Servicio REST. Esta invocación es implementada utilizando el componente *HTTPService* de *Flex*. La información

recibida como respuesta es utilizada para instanciar adecuadamente la clase *wrapper*. De esta manera, el acceso a la información es transparente desde el código de la IU.

Para obtener el código de integración, se proporciona una regla de generación que es aplicada sobre cada clase del Modelo de Objetos, que sea referenciada en el Modelo de Interacción Abstracto. Esta regla se basa en la plantilla de una clase *ActionScript* con una serie de operaciones predefinidas para consultar el estado de los objetos. Adicionalmente, la plantilla genera el código de invocación de aquellas operaciones del Modelo de Objetos que son utilizadas. Debido a que existe cierta similitud conceptual entre el concepto de clase de *ActionScript* y el del Modelo de Objetos, la generación es, prácticamente, una mera traducción sintáctica.

5.5 Conclusiones

En el presente capítulo se ha abordado, desde la perspectiva del DSDM, como generar una IU para una tecnología RIA. Para ilustrar la aproximación, se ha incorporado la propuesta en el método OOWS 2.0 y, se ha presentado la generación de la IU para la tecnología RIA *Adobe Flex*. Los objetivos marcados inicialmente han sido satisfechos mediante las siguientes contribuciones:

- La especificación de un metamodelo que captura la expresividad genérica de las interfaces RIA y que es especializado, en función de una tecnología concreta.

- La especificación de un metamodelo y una sintaxis textual, para la definición del comportamiento basado en eventos de una Interfaz RIA.

- La definición de una estrategia de conexión, basada en el concepto de *weaving*, entre los metamodelos propuestos y los metamodelos de un método de Ingeniería Web.

- La ilustración del uso de las contribuciones en el marco del método de Ingeniería Web OOWS 2.0, tanto a nivel de modelado como de generación de código.

El uso de modelos conceptuales permite el desarrollo de interfaces RIA, que aprovechen las características específicas de cada tecnología, pero sin añadir una complejidad excesiva al proceso de desarrollo. Además, como aplicación más relevante, estos modelos se han integrado en el método de Ingeniería Web OOWS 2.0 que también tiene en cuenta la generación de la funcionalidad. El objetivo de este capítulo, ha sido el de presentar un marco de modelado conceptual, a partir del cual es factible la generación de código. Sin embargo, la implementación industrial de las herramientas y el compilador de modelos asociado queda fuera del alcance de la presente tesis. Independientemente, gracias al análisis preciso con el cual se han definido los metamodelos, resulta factible el objetivo de la generación automática del código de la interfaz, porque la expresividad necesaria se encuentra presente a nivel de modelado conceptual. Para soportar esta afirmación, se han definido las líneas generales de un proceso de generación de código para la tecnología *Adobe Flex*, utilizando los modelos introducidos en el método OOWS 2.0. Como trabajo futuro queda pendiente llevar estas ideas a un marco industrial.

Capítulo 6

Patrones de Modelado para Aplicaciones Web 2.0

Los modelos conceptuales que proporcionan los métodos de Ingeniería Web cubren, en gran medida, parte de la expresividad necesaria para soportar este tipo de patrones habituales en las aplicaciones Web 2.0. Sin embargo, cada vez que uno de estos patrones es aplicado, tiene que crearse el conjunto de primitivas conceptuales correspondiente. Hasta ahora, pocos trabajos han abordado como introducir estas soluciones para su reutilización sistemática. En este capítulo introducimos el concepto de patrón Web 2.0, a nivel de modelado conceptual, para proporcionar modelos que encapsulen la expresividad descrita por estos patrones en el ámbito de los métodos de Ingeniería Web. Para definir con precisión el concepto, en primer lugar, en la sección 6.1 se analizan distintas aplicaciones Web 2.0 y repositorios de patrones, para determinar qué es un patrón Web 2.0. Como resultado de dicho análisis, se determina un conjunto relevante de patrones a modelar. A continuación, en la sección 6.2, se proporciona una aproximación basada en modelos para documentar los patrones Web 2.0. Con el fin de ejemplificar la propuesta, se describen detalladamente cinco patrones Web 2.0. En la sección 6.3, se define una aproximación genérica para introducir los patrones Web 2.0 en el ámbito de un método de ingeniería Web. A modo de ejemplo

se ilustra esta aproximación en el marco del método OOWS 2.0. Por último, se presentan las conclusiones en la sección 6.4.

6.1 Análisis de patrones Web 2.0

En los capítulos anteriores, se ha tratado el desarrollo de aplicaciones Web 2.0 desde el punto de vista de las tecnologías utilizadas para soportar el desarrollo. No obstante, no se debe olvidar la faceta social tan habitual en este tipo de aplicaciones. Un elemento diferenciador de las aplicaciones Web 2.0 es su énfasis en el usuario final. La consecuencia de este hecho es la aparición de un gran número de aplicaciones Web 2.0, cuya finalidad es la creación de comunidades de usuarios y facilitar la interacción social entre ellos. Es innegable la popularidad que han alcanzado estas aplicaciones de carácter social, hasta tal punto que *Facebook.com* ha superado, en los Estados Unidos (a fecha de Marzo del 2010), en cuota de mercado al buscador *Google.com*, según la consultora *Hitwise* [Hitwise 2010]. El principal modelo de negocio de estas aplicaciones sociales se basa en aglutinar un gran número de usuarios que aporten su información personal y a partir de la misma, definir campañas publicitarias a medida. Otro modelo de negocio habitual es la creación de una gran base de usuarios, con el objetivo de ganar popularidad en su segmento y, posteriormente, proporcionar servicios de pago.

Estas aplicaciones usan una serie de mecanismos a la hora de proporcionar la funcionalidad al usuario. De hecho, muchos de estos mecanismos se encuentran de modo recurrente, ya que simplifican la interacción del usuario con la aplicación, aspecto el cual es un factor decisivo en el ámbito de las aplicaciones sociales. Tal y como se describe en la sección 3.4 del estado del arte, muchos de estos mecanismos han sido documentados como patrones y recopilados en diversos repositorios creados a tal efecto ([Yahoo 2008], [Fraternali, Tisi *et al.* 2009] o [Toxboe 2009]). Dado que estos patrones han sido un factor clave en la implantación de las aplicaciones Web 2.0, es interesante introducir su uso en el ámbito de los métodos de Ingeniería Web.

Como paso previo a la definición del concepto de patrón Web 2.0, se ha realizado un análisis de diversos patrones. Con dicho fin, se ha seleccionado

un conjunto de catorce patrones que eran comunes a dichos trabajos y, que simplifican la interacción del usuario con la aplicación Web 2.0. Para diferenciar estos patrones de los que también estaban presentes en la "Web 1.0", la interacción tiene que tener como objetivo realizar alguna acción, o bien sobre el contenido de la aplicación, por ejemplo una descripción, una crítica o un objeto multimedia, o bien sobre los usuarios y sus conexiones sociales. En consecuencia, se han omitido aquellos patrones que se centraban en mejorar la apariencia visual o, en funcionalidad que no influía sobre el contenido o la faceta social de la aplicación.

Ya que muchos patrones Web 2.0 eran similares entre sí, pero con ligeras variaciones, se han agrupado aquellos patrones que proporcionaban una funcionalidad similar con el fin de presentar un conjunto más reducido. A continuación, se describe brevemente el propósito de los patrones seleccionados, junto con un ejemplo extraído de una aplicación Web 2.0 real:

1. **Quick Comment***:* este patrón es utilizado para que los usuarios introduzcan sus comentarios personales, sobre un contenido específico que se muestra en una página Web. El patrón se compone de un cuadro de texto embebido en la propia página que permite definir y enviar un comentario textual, tal y como muestra la Fig. 6.1. Cuando el comentario es enviado, automáticamente, la página es actualizada para que el usuario perciba que realmente ha sido añadido. Proporcionando un mecanismo sencillo, el patrón enfatiza que los usuarios introduzcan sus opiniones personales.

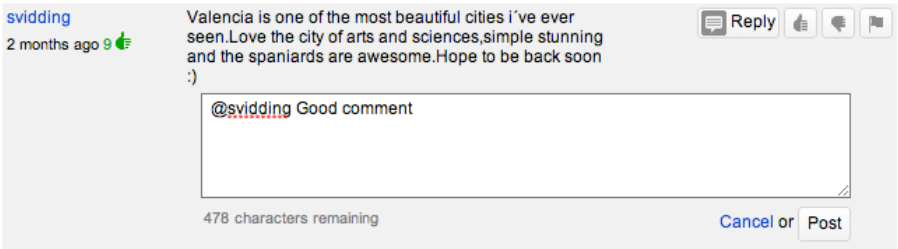


Fig. 6.1: Ejemplo del patrón *Quick comment*

2. ***Tag Definition:*** en el ámbito de la Web 2.0 es muy habitual el concepto de *tag* o etiqueta: palabras que se utilizan para categorizar el contenido de la aplicación. Estos *tags* son habitualmente creados por los distintos usuarios que visitan el sitio Web. Mediante este patrón se define la creación y asociación de dichos *tags* por parte del usuario. Se detectan dos usos de este patrón: (1) el usuario introduce un nuevo *tag* que describe el contenido que esta visualizando y (2) el usuario selecciona, a partir de un conjunto de *tags* predefinidos, cuales son los más adecuados. Ambos usos quedan reflejados en la Fig. 6.2 en la cual el usuario puede escribir nuevos *tags* o seleccionarlos de un conjunto propuesto por el sistema. El uso del patrón permite, además, la creación colaborativa de categorizaciones del contenido.

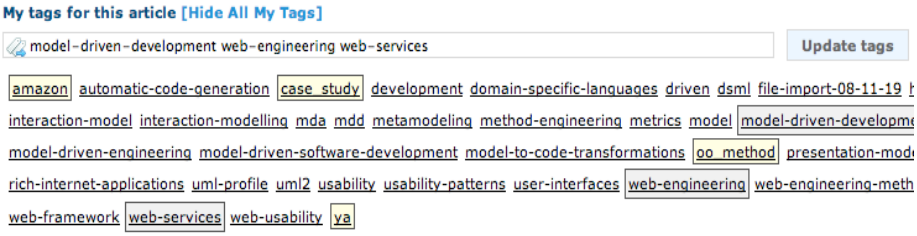


Fig. 6.2: Ejemplo del patrón *Tag definition*

3. **Notification***:* este patrón sirve para informar al usuario sobre eventos que se han producido en el ámbito de la aplicación, que puedan resultar de su interés. En el ámbito de las aplicaciones Web 2.0, las notificaciones son muy habituales para informar al usuario de cambios recientes sobre sus contactos o, de nueva información y funcionalidad disponible. Por ejemplo, en la Fig. 6.3 se muestra un conjunto de notificaciones que informan al usuario sobre su etiquetado en varias fotos y sobre un nuevo comentario a una contribución suya. El patrón resalta las nuevas notificaciones cada vez que el usuario se conecta manteniendo, además un registro de las más recientes.

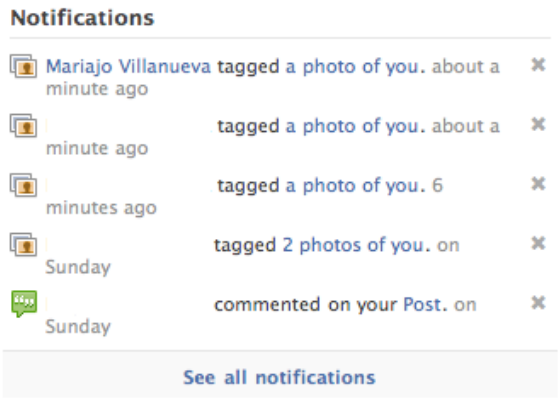


Fig. 6.3: Ejemplo del patrón *Notification*

4. **Collaborative Editing**: este patrón es utilizado para que el contenido de la página Web sea creado y modificado por los propios usuarios. El patrón consiste en la edición del contenido, mediante un editor disponible en la propia interfaz de la aplicación, tal y como muestra el ejemplo de la Fig. 6.4. Utilizando este editor cualquier usuario, registrado o anónimo según el perfil de autenticación seleccionado, puede realizar o añadir modificaciones. Todas estas modificaciones son almacenadas en un historial, de tal forma que los propios usuarios pueden consultar la evolución del contenido y discutir sobre que modificaciones debe ser incluidas. Este patrón es la base sobre la cual se sustentan los llamados *wikis*: aplicaciones Web 2.0 cuyo contenido es completamente definido por los propios usuarios.

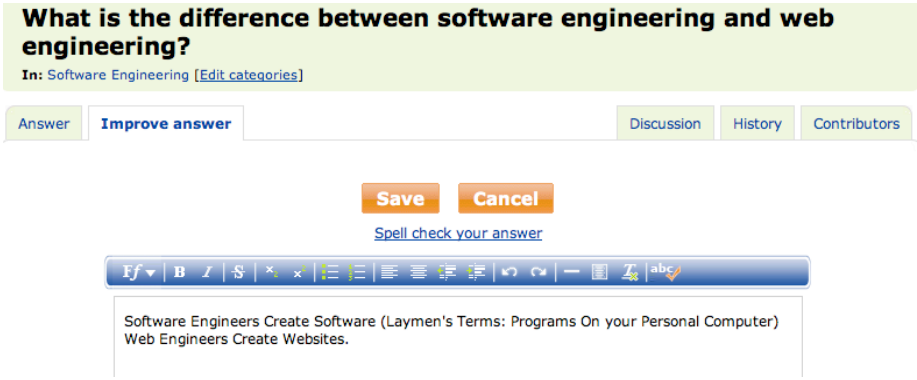


Fig. 6.4: Ejemplo del patrón *Collaborative Editing*

5. **Quick Rating**: este patrón simplifica la evaluación del contenido mostrado. Esta evaluación se realiza, o bien mediante una escala (Ver Fig. 6.5 izquierda) que otorga un valor numérico al contenido, o bien a través de un escala cualitativa predefinida, como si el contenido le gusta o no al usuario (Ver Fig. 6.5 derecha). El uso del patrón evalúa el contenido mediante una única interacción y establece, directamente, cual es el contenido preferido por los usuarios de la aplicación.

Fig. 6.5: Ejemplos del patrón *Quick Rating*

6. **Reputation**: este patrón es utilizado para evaluar a los distintos usuarios que hacen uso de la aplicación. Los mismos usuarios se encargan de establecer la reputación de otros miembros de la comunidad, en función de su participación o las acciones que han realizado. Este patrón es muy útil para establecer relaciones de confianza sobre todo, cuando solo existe un contacto virtual entre los usuarios. Por ejemplo, la Fig. 6.6 muestra un valor numérico que determina la reputación del usuario, basándose en los distintos votos, positivos o negativos, que le han concedido el resto. El uso del patrón permite discernir, además, que usuarios han aportado contenidos de mayor calidad a la aplicación. A diferencia del patrón *Quick Rating*, que se apli-

ca sobre el contenido, este patrón se aplica sobre los propios usuarios de la aplicación. Sin embargo, existe una estrecha relación entre ambos patrones. Puesto que es muy habitual que el contenido se encuentre enlazado con el usuario que lo creó, la reputación del usuario puede derivarse de manera implícita a partir de la evaluación del contenido aportado.

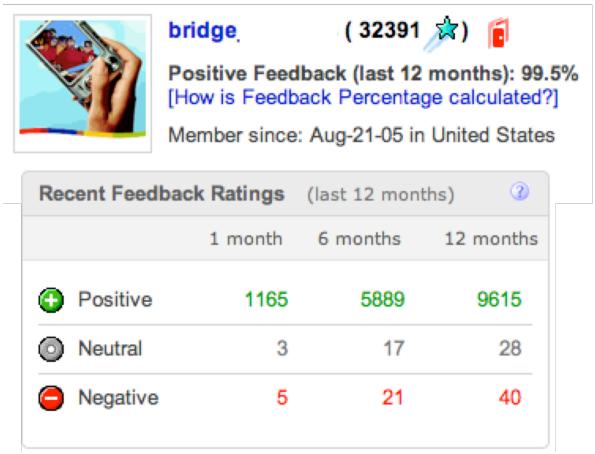


Fig. 6.6: Ejemplo del patrón *Reputation*

7. **Share Content**: el uso de este patrón permite compartir el contenido disponible en una aplicación Web con otras aplicaciones. La implementación habitual del patrón consiste en un conjunto de enlaces, cada uno de los cuales representa una aplicación Web. Cuando el usuario selecciona uno de estos enlaces, el contenido es publicado automáticamente en la aplicación seleccionada. Por ejemplo, cuando un usuario ve un video interesante, mediante el uso de este patrón, puede publicarlo instantáneamente en su *blog* personal. Como paso previo, es habitual que se solicite la identificación del usuario para poder publicar el contenido en la aplicación destino. Gracias a este patrón, se facilita la difusión del contenido en el ámbito de la Web 2.0.

Fig. 6.7: Ejemplo del patrón *Share Content*

8. **Suggestion**: la aplicación de este patrón sugiere al usuario contenido u otros usuarios considerando sus preferencias. En base a la información que se dispone del usuario, como su contenido preferido o su lista de contactos, el sistema le proporciona un conjunto de contenidos que, por sus características, pueden resultar de su interés. Desde la perspectiva social, la aplicación de este patrón facilita que el usuario pueda ampliar su círculo de contactos.

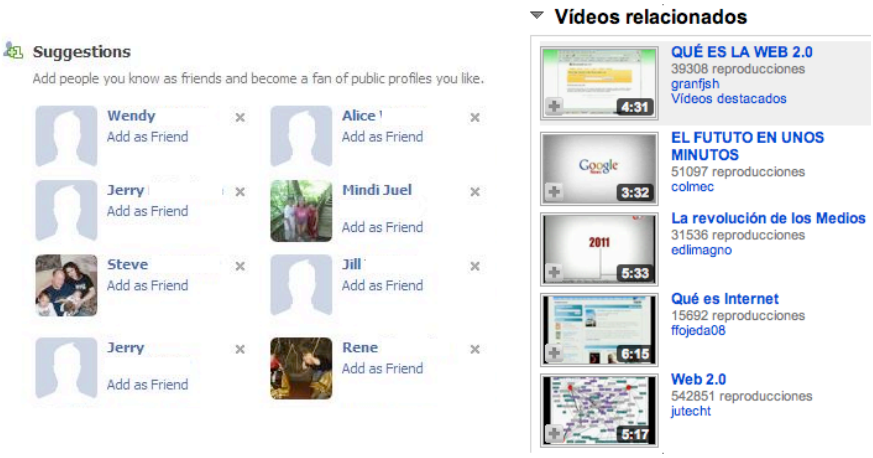


Fig. 6.8: Ejemplos del patrón *Suggestion*

9. **Invite**: este patrón es utilizado para que un usuario realice recomendaciones personales a su círculo de contactos. Entre las recomendaciones más habituales en el ámbito de las aplicaciones Web 2.0, se encuentran unirse a un grupo específico, probar una nueva funcionalidad o la participación en un evento. La interacción habitual del patrón es la de proporcionar una lista de los contactos, con el objeti-

vo que el usuario seleccione a quienes enviar la invitación (ver Fig. 6.9). Una vez recibida la invitación, ésta es aceptada o rechazada mediante una interacción sencilla simplificando, de este modo, el proceso de confirmación. Muchas aplicaciones Web 2.0 incluyen este patrón en su proceso de registro, para que los nuevos usuarios recomienden la propia aplicación a sus contactos. La principal ventaja del patrón reside en que la invitación sea realizada de forma personal, ya que de esta manera, aumentan las posibilidades de captar nuevos usuarios.

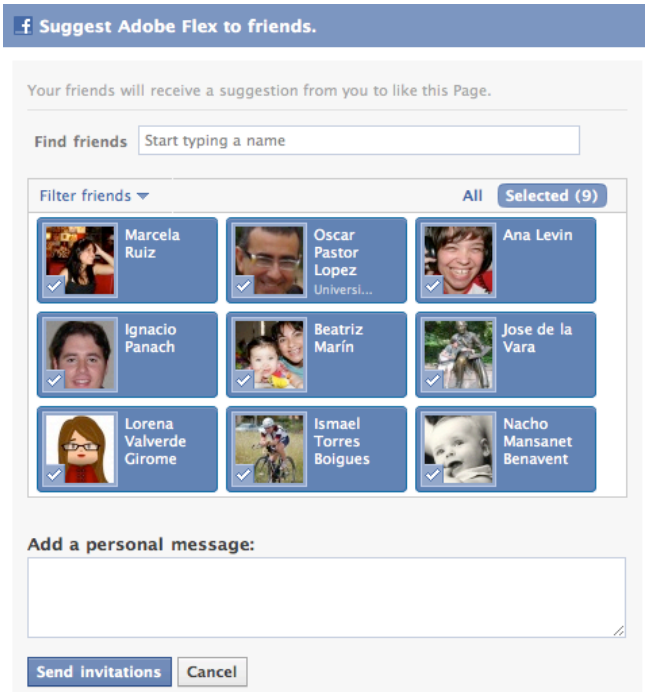


Fig. 6.9: Ejemplo del patrón *Invite*

10. **Public Profile**: mediante el uso de este patrón se genera un resumen sobre la información del usuario. A diferencia del perfil del usuario de la aplicación, este patrón genera una de tarjeta de identificación con un conjunto de información básica o acotada por el propio usuario, que es visible a través de una URL pública. En consecuencia, este perfil es utilizado por las herramientas de búsqueda, tanto internas

como externas de la aplicación. Su principal función es que el usuario de a conocerse en la Web, pero manteniendo parte de su privacidad.

Fig. 6.10: Ejemplo del patrón *Public Profile*

11. **Availability**: este patrón informa sobre los usuarios que actualmente se encuentran utilizando la aplicación. Su implementación habitual se basa en una lista de los usuarios conectados a partir de la cual es posible acceder a su perfil (Ver Fig. 6.11). El uso de este patrón adquiere sentido si la aplicación incluye un mecanismo de comunicación entre usuarios, como por ejemplo un *chat*. De este modo, se fomenta la interacción entre usuarios que se encuentran en un momento determinado usando la aplicación, sin necesidad de una herramienta externa de mensajería instantánea. También es común el uso de este patrón para establecer nuevos contactos.

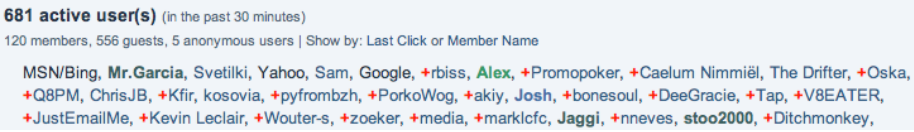


Fig. 6.11: Ejemplo del patrón *Availability*

12. **Ranking***:* la aplicación de este patrón otorga un rango a cada usuario, el cual se va modificando conforme se alcanzan unos objetivos específicos en el marco de la aplicación. Por ejemplo (ver Fig. 6.12),

cuando un usuario consigue un número elevado de comentarios, se le otorga un rango mayor que destaca su grado de colaboración. Este patrón establece una clasificación o *ranking* de los usuarios, el cual es visible para el resto de la comunidad. A diferencia del patrón *Reputation,* el rango del usuario se alcanza, no por la decisión del resto de la comunidad, sino de forma automática y predefinida. Además, la funcionalidad disponible puede asociarse al rango que ostenta cada usuario. De esta manera, el patrón permite la restricción de las acciones que realizan los nuevos usuarios en la aplicación, hasta que no alcanzan un cierto nivel de colaboración.

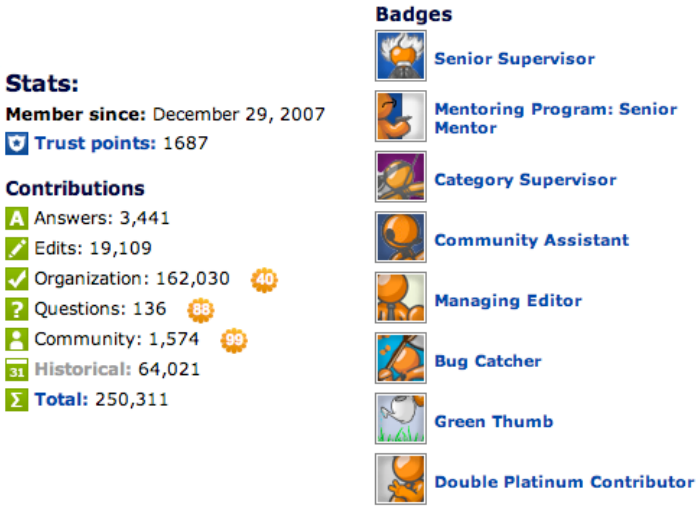


Fig. 6.12: Ejemplo del patrón *Ranking*

13. **Favorites**: este patrón es utilizado para marcar el contenido favorito del usuario. El mecanismo de interacción habitual de este patrón se basa en un icono, por ejemplo, una estrella como ilustra la Fig. 6.13, que permite seleccionar un contenido concreto como favorito. Una vez marcado, los contenidos favoritos son listados desde el perfil del propio usuario. La aplicación de este patrón resulta útil para determinar cual es el contenido de mayor aceptación entre los usuarios de la aplicación.

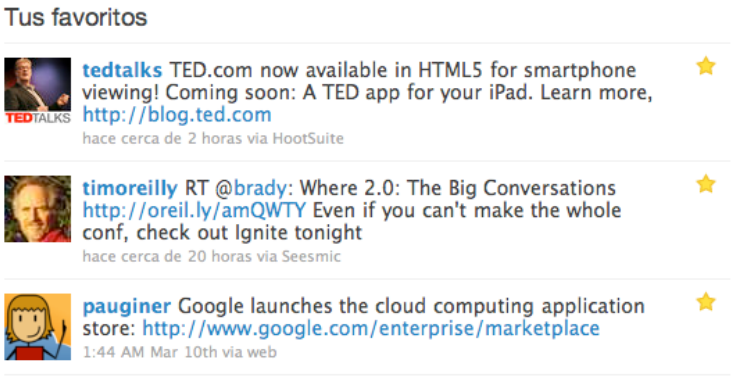


Fig. 6.13: Ejemplo del patrón *Favorite*

14. **Subscription**: este patrón define un mecanismo mediante el cual, un usuario se subscribe a los cambios que se producen en algún contenido o sobre un usuario. Básicamente, el patrón proporciona algún tipo de enlace, por ejemplo, el botón "seguir" tal y como muestra la Fig. 6.14, mediante el cual el usuario se subscribe automáticamente. Cuando se produce algún cambio, como por ejemplo la creación de un nuevo contenido, estos cambios son notificados automáticamente a todos los usuarios suscritos. El uso de este patrón establece una relación de fidelidad entre los usuarios que, además, determina indirectamente el contenido y los usuarios que tienen una mayor aceptación en la comunidad. A diferencia del patrón *Notification,* en este patrón es el propio usuario quién decide qué notificaciones recibe mientras que, en el caso anterior, la decisión recae en la aplicación. El patrón también proporciona, habitualmente, una visualización que únicamente muestra los cambios producidos en los contenidos o usuarios sobre los cuales está suscrito.

Fig. 6.14: Ejemplo del patrón *Subscription*

Una vez descritos los patrones, el siguiente paso ha sido el de analizar cuales eran más populares en base a su utilización real en la implementación de aplicaciones Web 2.0. Para seleccionar el conjunto de páginas Web a analizar, se ha utilizado el índice de tráfico proporcionado por la organización *Alexa* [Alexa 2010]. Esta organización recopila información del uso de la distintas aplicaciones Web mediante la monitorización de sus visitas. A partir de la información recopilada, se elabora una clasificación con los sitios con un mayor número de tráfico. Para realizar el análisis, se han seleccionado un conjunto de aplicaciones Web 2.0 que ocupaban una posición relevante en el *top* 500. Con el objetivo de obtener una muestra más representativa, se han omitido las variantes regionales de algunas páginas. También se han descartado páginas pertenecientes a dominios de aplicación muy similares para obtener una muestra más variada. Por ejemplo, con el auge de las redes sociales han surgido multitud de sitios clónicos que son similares a *facebook*. En consecuencia, es de esperar que los patrones aplicados sean los mismos. Por dicha razón, se ha optado por utilizar aplicaciones Web 2.0 pertenecientes a dominios diferentes.

En total se han seleccionado quince aplicaciones o sitios Web determinando, para cada una de ellos, si un patrón concreto se aplicaba o no. Se ha añadido al conjunto de aplicaciones el ejemplo ilustrativo de *My Social Research* presentado en el Anexo A.1. Respecto al análisis, cabe reseñar dos condiciones: (1) el análisis se ha realizado a fecha de Marzo de 2010, con lo cual es de esperar que con el paso del tiempo nuevos patrones sean soportados o descartados; (2) la implementación del patrón en cada sitio no era idéntica completamente. Se ha considerado el patrón como soportado, si el sitio proporcionaba un mecanismo con el mismo propósito que el patrón. La Tabla

6.1 muestra para cada patrón, si ha sido aplicado (con una letra A) o no (con una letra X). Por otro lado la Fig. 6.15, resume el porcentaje de aplicación total de cada patrón en los sitios seleccionados.

Tabla 6.1: Relación entre los patrones y las webs analizadas

Aplicación Web 2.0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
facebook	A	X	A	A	A	X	X	A	A	A	A	A	X	A
youtube	A	X	A	A	A	X	A	A	X	A	X	X	A	A
wikipedia	A	A	A	A	X	A	X	A	X	A	X	A	X	A
blogger	A	A	A	A	X	X	A	X	A	A	X	X	A	A
myspace	A	X	X	X	A	X	X	A	A	A	A	X	A	X
twitter	A	X	A	X	X	A	A	A	A	A	A	X	A	A
ebay	A	A	A	X	A	A	A	A	X	A	X	X	X	A
amazon	A	X	X	A	A	A	X	A	X	X	X	X	A	X
flickr	A	A	A	X	X	X	A	X	A	A	X	X	A	X
linkedin	A	A	A	X	A	A	X	A	A	A	X	X	X	A
orkut	A	X	A	A	A	X	A	A	A	A	A	X	A	A
livejournal	A	A	A	X	X	X	A	A	A	A	X	X	A	A
urban dictionary	A	A	X	A	A	A	A	A	X	A	A	X	A	X
digg	A	A	X	X	A	A	A	A	X	A	A	A	X	A
answers	A	A	X	A	A	A	A	A	X	A	X	A	A	A
my Social Research	A	A	A	A	A	A	A	A	X	A	X	X	A	A

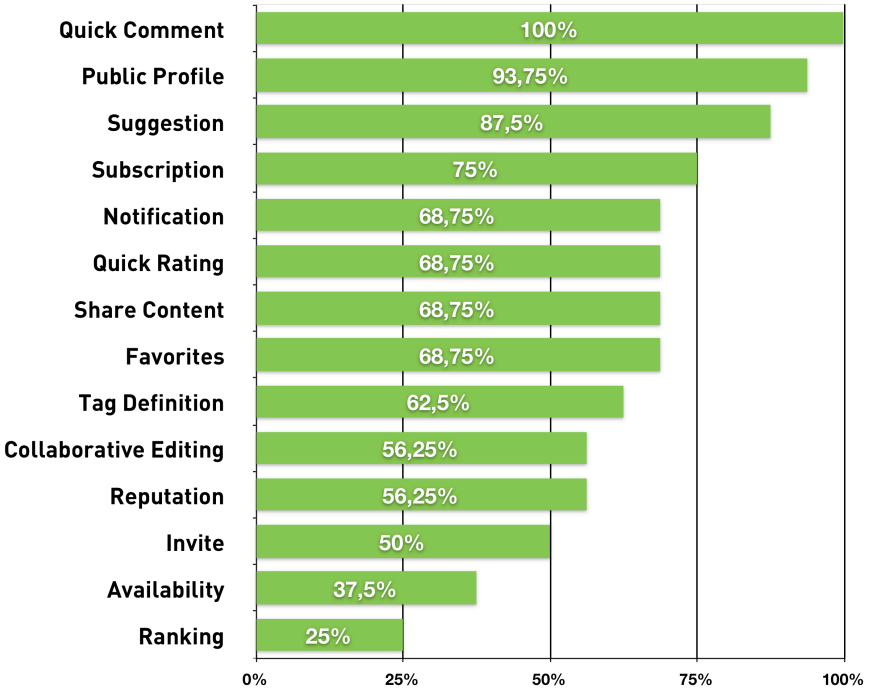


Fig. 6.15: Porcentaje de aplicación de cada patrón

Analizando los resultados se extraen las siguientes conclusiones:

- Un total de doce patrones son aplicados en un porcentaje mayor al 50%, mientras que solo dos (*Availability* y *Ranking*) se consideran poco relevantes al tener un porcentaje inferior al 40%. También cabe destacar que muchos de estos patrones se encuentran implementados de modo casi idéntico, siendo fácilmente identificables. En consecuencia, varios patrones son claros candidatos a ser soportados sistemáticamente en un método de Ingeniería Web.

- El patrón *Quick Comment* es aplicado en el 100% de las aplicaciones Web 2.0 analizadas. El hecho que el usuario introduzca su opinión personal se convierte, prácticamente, en una funcionalidad estándar en las aplicaciones Web 2.0. Este patrón simplifica en gran medida

dicha interacción. El segundo patrón más utilizado, *Public Profile*, alcanza un 94%. En todas las aplicaciones analizadas, existe la posibilidad de registrarse, de ahí que la creación de un perfil público fuese directa.

- Destaca el bajo porcentaje del patrón *Ranking*. Si bien la posibilidad de evaluar a los usuarios se encuentra ampliamente difundida, tal y como muestra el patrón *Reputation* con un 69% de aplicación, no ocurre lo mismo en la categorización de los usuarios. Algunos trabajos destacan, [Crumlish & Malone 2009], que el hecho de crear "élites" de usuarios es contraproducente para el objetivo social de la aplicación Web 2.0. El patrón *Availability* tampoco consigue un resultado elevado (38% de aplicación), puesto que únicamente es utilizado en aquellas aplicaciones que disponen de un cliente de mensajería instantánea integrado. Cabe señalar que ambos patrones son habituales en los foros de discusión, sin embargo, en este análisis no se ha considerado ninguno.

- En conjunto, los patrones relacionados con mejorar la interacción con el contenido obtienen un mejor resultado global, que los patrones relacionados con fomentar la interacción social. A pesar del carácter social de la Web 2.0, exceptuando las aplicaciones basadas en redes sociales, todavía no se proporciona un soporte avanzado a la creación de comunidades en otros dominios.

6.2 Patrones Web 2.0: modelado conceptual

Una vez realizado el análisis, el siguiente paso es la introducción de los patrones en un método dirigido por modelos. A diferencia de los patrones que se presentaban en la sección 4.3.3, los cuales se centraban en el desarrollo de la interacción, estos patrones están enfocados a destacar una serie de ventajas que proporcionan al usuario que utiliza la aplicación. Es decir, los patrones del Modelo de Interacción Abstracto modelan una implementación de una interacción que se producía habitualmente. En cambio, estos patrones son utilizados para mejorar la experiencia del usuario a la hora de realizar dichas

interacciones con la aplicación. Por lo tanto, definimos el concepto de patrón Web 2.0, como la abstracción de un mecanismo de interacción en el ámbito de las aplicaciones Web 2.0, con el fin de mejorar la experiencia del usuario con la aplicación.

La mayoría de los repositorios de patrones analizados en el Estado del Arte (ver sección 3.4) documentan los patrones utilizando una descripción textual de su propósito. En algunos casos, se aporta una implementación del mismo, o bien basada en un ejemplo de aplicación real, o bien mediante el propio código que implementa el patrón. En estos repositorios se encuentra el razonamiento del patrón, el porqué de su uso y opcionalmente, una posible implementación. Al igual que en muchas aproximaciones basadas en patrones, estos trabajos utilizan una plantilla textual para realizar una descripción detallada utilizando distintos apartados. Esta parte descriptiva del patrón es relevante puesto que captura su semántica. Cabe señalar que en el marco de los patrones Web 2.0, se hace más hincapié en el aspecto social que enfatizan, que en cómo abordar su implementación. Por dicha razón, la plantilla utilizada difiere ligeramente de la discutida en la sección 4.3.3. En particular, en esta tesis doctoral se toman como base algunos apartados que utilizan dichos repositorios, a fin de definir nuestra propia plantilla de descripción de patrones Web 2.0. La plantilla propuesta se compone de los siguientes apartados:

- *Problema*: descripción breve del problema que motiva el uso del patrón.

- *Rationale*: razonamiento que explica el porqué la aplicación del patrón es útil, para mejorar experiencia del usuario en la interacción con la aplicación Web 2.0. Además, se incluyen las ventajas que aporta el patrón sobre otros mecanismos utilizados con el mismo propósito.

- *Contextos de uso*: escenarios o situaciones, en el marco de una aplicación Web 2.0, en los cuales se recomienda el uso del patrón. Opcionalmente, también puede documentarse en que casos no es recomendable o es contraproducente aplicar el patrón.

- *Solución*: descripción textual que detalla cómo el patrón resuelve el problema que se ha definido. Este apartado puede incluir detalles específicos de la implementación que faciliten la comprensión de la solución. Si existen soluciones alternativas, también son descritas en este apartado.

- *Ejemplo*: un ejemplo de una aplicación Web 2.0 real en la cual se ha utilizado el patrón.

En la presente tesis se aborda no solo la documentación de los patrones, sino también su posterior introducción en un método de Ingeniería Web dirigido por modelos. No obstante, a partir de las descripciones textuales esta labor no resulta sencilla. En primer lugar, una descripción textual da lugar a múltiples ambigüedades e interpretaciones, de tal forma que la implementación final que se realiza del patrón puede distar de la idea original que se pretende representar. En segundo lugar, en la mayoría de los métodos de Ingeniería Web, las implementaciones se especifican en base a modelos conceptuales, por lo que las descripciones textuales tienen que ser "transformadas" a dichos modelos.

Debido a ambos inconvenientes, en esta tesis se propone la definición de los patrones Web 2.0, no solo como una mera descripción sino también desde un punto conceptual, es decir, introduciendo modelos conceptuales que describan su semántica. La utilización de modelos conceptuales para caracterizar los patrones permite resolver la ambigüedad, hasta cierto punto, ya que la solución tiene que ser fiel a dichos modelos. Con el objetivo de proporcionar una solución genérica, estos modelos tienen que cumplir dos requisitos fundamentales. En primer lugar, tienen que ser independientes de un método de Ingeniería Web o plataforma tecnológica específica. El objetivo de establecer el uso de patrones es, precisamente, la aplicación de la solución en distintos métodos. Por esta razón, la genericidad de la aproximación tiene que ser preservada. En segundo lugar, los modelos tienen que ser descritos mediante una notación que sea sencilla y comprensible para la mayoría de los analistas de software. No es necesario reinventar notaciones para este propósito. La reutilización de otras notaciones, ampliamente aceptadas por la comunidad de la Ingeniería del Software, es una solución apropiada con el fin de definir formalmente la semántica del patrón.

En el marco de la presente tesis doctoral, se han seleccionado dos aspectos de la semántica del patrón que son representados mediante modelos conceptuales: funcionalidad e interacción. Por un lado, cada patrón establece un conjunto de funcionalidad e información a introducir en la aplicación. Por ejemplo, qué datos del usuario son utilizados o qué servicio se ejecuta para realizar las valoraciones del contenido. Por otro lado, cada patrón determina también el flujo o secuencia de acciones que realiza el usuario. Cada uno de estos aspectos es representado mediante un modelo conceptual, que es incluido en la descripción del patrón. Los modelos propuestos son:

- **Modelo de funcionalidad**: con el propósito de abstraer la funcionalidad que representa un patrón se ha seleccionado el Diagrama de Clases de UML. La razón que justifica la selección es el hecho de tratarse de una de las notaciones más utilizadas, en el ámbito de la Ingeniería Web. Además diversos trabajos basados en patrones, como por ejemplo el trabajo original propuesto por [Gamma, Helm *et al.* 1995], utiliza este modelo con el mismo objetivo. Varios métodos de Ingeniería Web ofrecen modelos que guardan una estrecha relación semántica con el Diagrama de Clases de UML, de tal modo que se simplifica la tarea de trasladar la semántica del patrón al método. Aplicando esta aproximación, cada clase representa, mediante los atributos y las distintas asociaciones, las necesidades de información del patrón mientras que a través de las operaciones, se especifica la funcionalidad necesaria.

- **Modelo de interacción**: para representar la interacción que define el patrón, se ha utilizado la notación conocida como *ConcurTaskTrees* (CTT) [Paternò, Mancini *et al.* 1997]. Esta notación ha sido ampliamente utilizada, para el modelado de tareas, en entornos de desarrollo de interfaces dirigidos por modelos. La notación representa una estructura jerárquica, en forma de árbol, donde los nodos corresponden a las distintas tareas que realiza el usuario en la interacción con el sistema. Se distinguen tres tipos de tareas: (1) *Abstract Tasks*, que se componen asimismo de otras tareas; (2) *Interactive Tasks,* que representan una tarea que incluye la interacción del usuario con el sistema; (3) *System Tasks,* que son tareas que realiza el sistema de información sin ninguna intervención del usuario. Las

distintas tareas se interrelacionan entre sí a través de un conjunto de operadores que establecen sus relaciones temporales. Los operadores que proporciona la notación CTT se resumen en la Tabla 6.2. Utilizando dichos operadores, esta notación define, de un modo visual y sencillo, la interacción representada por los patrones estableciendo un orden claro de tareas. En esta tesis, se ha decidido adaptar la notación gráfica original con el fin de mejorar la legibilidad, al igual que han hecho previamente otros autores como [Valderas 2008] o [Nóbrega, Nunes *et al.* 2006]. Por lo tanto, tomando como base la notación de Paternò, los distintos nodos del árbol han sido representados mediante una forma gráfica específica, en función del tipo de tarea que representan. Adicionalmente, los arcos entre tareas han sido etiquetados mediante los operadores que proporciona la notación.

Tabla 6.2: Notación de tareas basada en CTT

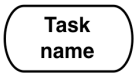
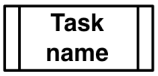
Tipo de tarea	Forma gráfica
Abstract	
Interactive	
Application	

Tabla 6.3: Operadores de la notación CTT

Operador	Símbolo	Descripción
Independent Concurrency	T1 ||| T2	Las tareas T1 y T2 se realizan en cualquier orden sin ningún tipo de restricción
Choice	T1 [ ] T2	Permite seleccionar una tarea, T1 o T2, de tal forma que la tarea descartada no puede realizarse, hasta que se finalice la tarea escogida
Concurrency with information exchange	T1 | [] | T2	T1 y T2 se ejecutan concurrentemente, pero tienen que sincronizarse para intercambiar información
Order Independence	T1 | = | T2	Se tienen que realizar las tareas T1 y T2, pero cuando una ha comenzado debe ser finalizada antes de comenzar la siguiente
Deactivation	T1 [> T2	La tarea T1 es desactivada una vez que la tarea T2 ha sido completada
Enabling	T1 >> T2	Cuando se finaliza la tarea T1, se permite la realización de la tarea T2
Enabling with information passing	T1 []>> T2	Ídem que el operador anterior pero, en este caso, la tarea T1 proporciona información a la tarea T2
Suspend- Resume	T1 |> T2	La realización de T2 interrumpe momentáneamente la realización de T1. Cuando finaliza T2, se continúa T1 en el estado en el cual se encontrase
Iteration	T1*	La tarea T1 se realiza todas las veces que el usuario considere necesario
Finite Iteration	$T1^{(n)}$	La tarea T1 puede repetirse n-veces

Cabe reseñar que otros modelos son perfectamente válidos, con el propósito de representar los aspectos de funcionalidad e interacción que proporcionan los patrones. La elección de la presente tesis doctoral se justifica en el hecho que son notaciones ampliamente conocidas, e independientes de un

método específico de Ingeniería Web. Además, existen diversas herramientas gráficas que soportan el uso de este tipo de notaciones.

Otro hecho a considerar es que la semántica de un patrón puede ser representada por múltiples modelos. En otras palabras, existen diversos modelos que, con ligeras variaciones, representan la semántica del patrón *Quick Rating*. Nuestro objetivo no es el de modelar todas las posibles soluciones que representa dicho patrón, sino ofrecer modelos que garanticen una solución válida, que cubra el mayor número de escenarios donde es necesario aplicar el patrón. A continuación, se aplica la plantilla y las notaciones de modelado propuestas a los cinco patrones Web 2.0, con mayor porcentaje de aplicación en el análisis realizado.

6.2.1 Quick Comment

Problema: los usuarios tienen que introducir su opinión sobre los distintos contenidos de la aplicación mediante un proceso sencillo.

Rationale: este patrón se utiliza con gran asiduidad en el marco de las aplicaciones Web 2.0, ya que su principal objetivo es captar la opinión de los usuarios. La principal ventaja del patrón es que ofrece la posibilidad de realizar comentarios en la misma página sobre la cual se muestra el contenido, sin la necesidad de realizar un proceso complejo que provoque el desistimiento del usuario. La inmediatez en la percepción del resultado es otra característica determinante que proporciona este patrón, puesto que una vez el usuario introduce su comentario, éste es añadido a la página. Mediante la sencillez de la interacción, se fomenta significativamente la participación, hecho que explica la amplia difusión de este patrón.

Contextos de uso: en las aplicaciones Web 2.0 es habitual la presencia de contenidos tales como imágenes, opiniones o videos. Una forma de proporcionar valor añadido a estos contenidos es el de facilitar a los usuarios, la posibilidad de expresar su opinión mediante un comentario textual. De manera indirecta, los comentarios personales asociados al contenido proporcionan una visión general de la valoración de los usuarios. El patrón también resulta útil, en otros contextos, para que el propio usuario introduzca de forma sencilla aspectos sobre su vida diaria, que reciben el nombre de *twetts* en la jerga

de la Web 2.0. Ejemplos de estos comentarios son una descripción de la actividad que se encuentra realizando o una noticia de carácter personal. Este patrón, por consiguiente, también da soporte indirectamente a la actividad denominada *microblogging*.

Solución: la solución más habitual es la definición de un componente de entrada de texto, que se incluye en la parte inferior del contenido a ser comentado. Utilizando un botón, el usuario envía su opinión siempre y cuando, se haya registrado previamente en la aplicación o se acepten comentarios anónimos. Si el usuario se ha identificado, el comentario incluye su identificación e información adicional como su apodo. Una vez enviado el comentario, éste aparece de manera destacada en el listado de comentarios asociados al contenido. Implementaciones más avanzadas del patrón ofrecen componentes más elaborados para definir texto con formato (estilo de fuente, inclusión de enlaces, etc.) o, incorporar las implementaciones de otros patrones relacionados como *Quick Rating*.

Ejemplo: la Fig. 6.1 muestra una implementación del patrón extraída del sitio Web *YouTube*. El patrón *Quick Comment* se representa mediante una caja de texto que es activada cuando el usuario escribe sobre ella. Simultáneamente a la escritura del usuario, se activa el botón "*Post*" para enviar el comentario. La implementación mostrada permite tanto la respuesta como la evaluación de los comentarios introducidos, previamente, por otros usuarios.

Descripción de los modelos: la Fig. 6.16 muestra los modelos asociados al patrón, el cual se define mediante tres clases que representan, respectivamente: el contenido comentado, el conjunto de comentarios asociados al contenido y el usuario que ha realizado el comentario. El modelo de funcionalidad propuesto no solo soporta la creación de comentarios (operación *postNewComment*) sino que, además, habilita la posibilidad de editarlos (operación *edit*), crear respuestas sobre los mismos (operación *replyTo*) y subscribirse (operación *subscribe*) para recibir dichas respuestas automáticamente.

Respecto al modelo de interacción, en primer lugar, se establece una relación de activación que implica que el comentario tiene que ser creado (tarea *Create Comment*), antes de realizar otras operaciones sobre él. Estas operaciones, que pueden repetirse indefinidamente, se representan mediante la ta-

rea abstracta *Comment Operations*. Las tareas *Reply Comment* y *Edition* permiten al usuario realizar diversas modificaciones al comentario, que son almacenadas mediante la tarea de sistema *Store comment modifications*. Respecto a la tarea *Subscription*, una vez el usuario habilita el mecanismo de subscripción, se realizan dos tareas en el sistema, una para almacenar dicha subscripción y otra, para enviar la información relacionada con el comentario suscrito.

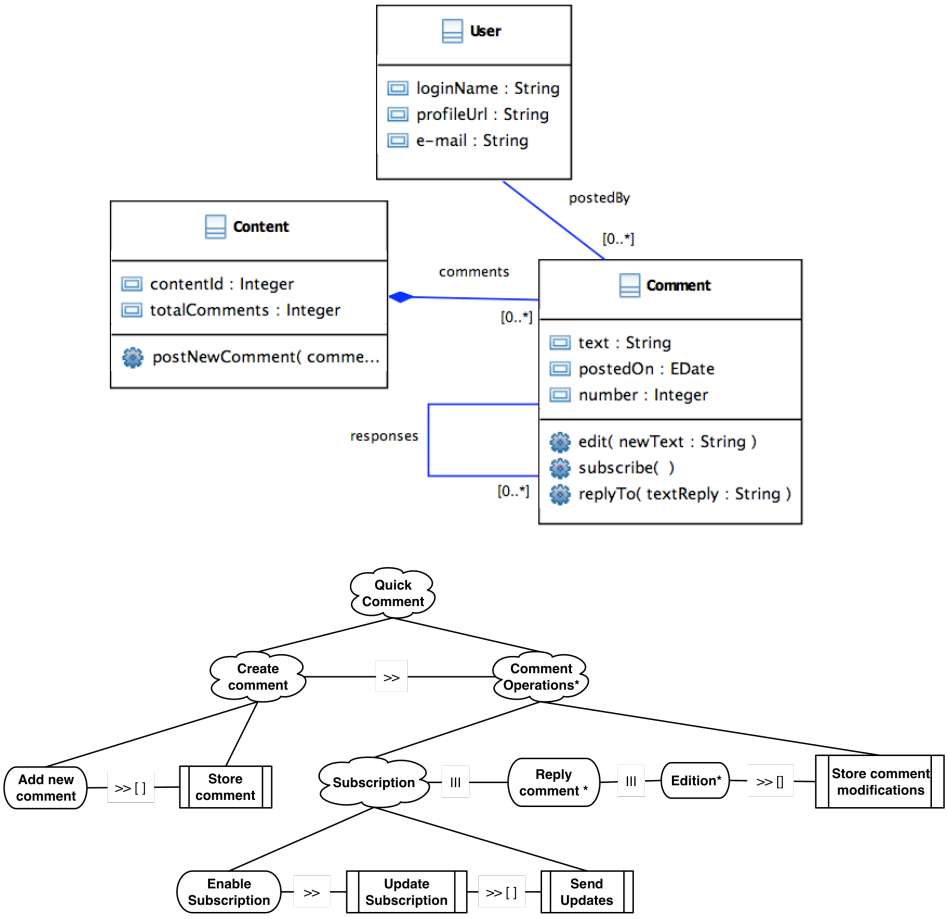


Fig. 6.16: Modelos del patrón *Quick Comment*

6.2.2 Notification

Problema: informar al usuario directamente de los cambios que se han producido en el marco de la aplicación.

Rationale: en el contexto de las aplicaciones Web 2.0, resulta fundamental que el usuario se encuentre constantemente informado de la novedades de la aplicación, así como de las aportaciones que realizan los distintos contactos de su red social. De esta forma se consigue un objetivo fundamental de muchas aplicaciones sociales: mantener el vínculo social entre los usuarios que no se lleva a cabo con asiduidad en el mundo real. El uso de este patrón permite que el usuario sea automáticamente informado, sin interacción implícita por su parte, de los cambios producidos en sus redes sociales. Este mecanismo favorece tanto el conocimiento como la respuesta a los mismos, hecho que, en definitiva, refuerza dicho vínculo social. Además, la implementación habitual del patrón resalta en la interfaz las nuevas notificaciones, ayudando a que el usuario les otorgue una mayor relevancia. Otro objetivo del patrón es mostrar al usuario información y funcionalidad, que pueda resultar de su interés. De esta manera, el usuario percibe la utilidad de la aplicación.

Contexto de uso: el patrón es útil en aquellas aplicaciones en las cuales ocurren una gran cantidad cambios y, por lo tanto, el usuario no puede estar pendiente de todos ellos. En el contexto de las redes sociales, cada interacción que realiza un contacto, potencialmente, genera un nuevo contenido o comentario. Si un usuario tiene una gran cantidad de contactos, es necesario que pueda obtener de forma sencilla estos cambios. El patrón también resulta útil cuando se desea que el usuario sea informado de eventos, que ocurren en el ámbito de la aplicación, pero que el mismo desconoce su existencia. No obstante, si la cantidad de notificaciones pendientes de consultar es muy elevada, el patrón resulta molesto para el usuario. Para evitar este inconveniente, en estos escenarios, deben limitarse el número de notificaciones destacando, únicamente, las más relevantes o filtrando las más antiguas.

Solución: la implementación habitual de este patrón es un componente de interfaz, habitualmente una lista de elementos textuales, que muestra el número de notificaciones pendientes de consultar y destaca las nuevas notificaciones recibidas. Algunas implementaciones únicamente muestran el

número de notificaciones, mientras que otras destacan, brevemente, una descripción sobre el cambio que ha producido la notificación. Independientemente, el patrón también ofrece los enlaces necesarios para acceder a la información detallada de la notificación. Una vez consultada la notificación o cuando ha transcurrido un tiempo recomendado, ésta es desactivada. Adicionalmente, la gran mayoría de las implementaciones incluyen la consulta al historial de notificaciones.

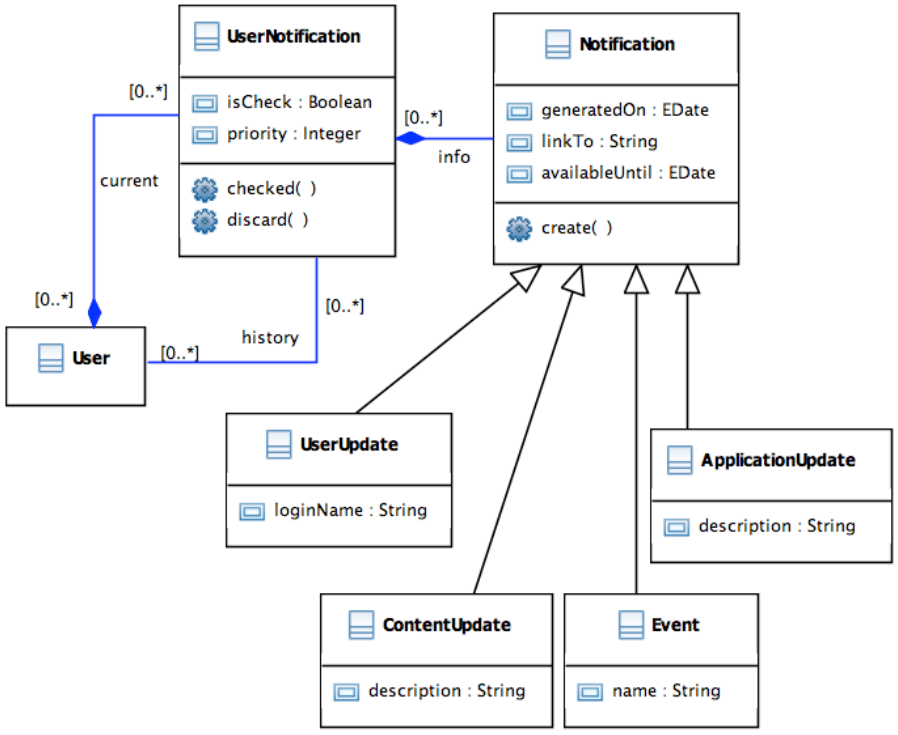
Por defecto, es la aplicación la encargada de seleccionar el conjunto de notificaciones a mostrar al usuario como por ejemplo, los últimos comentarios que han realizado otros usuarios en su perfil. Sin embargo, es frecuente que el usuario configure qué tipo de notificaciones desea recibir o no, incluso si desea ser informado, a través de correo electrónico, de las mismas.

Ejemplo: la Fig. 6.3 muestra la implementación del patrón *Notification* en la aplicación Web *Facebook*. Cuando se produce un evento relevante para el usuario en el contexto de la aplicación, en el ejemplo una serie de comentarios realizados sobre una foto y la respuesta a un comentario introducido por un usuario, dicho cambio es mostrado en el listado de notificaciones. Los elementos de dicha lista aparecen resaltados, cada vez que el usuario inicia sesión en *Facebook*, proporcionando una breve descripción y un enlace para ver la información asociada a la notificación. Las notificaciones consultadas dejan de estar resaltadas automáticamente. También es posible descartar una notificación mediante el botón, en forma de aspa, asociado a cada una de ellas. Aunque no se muestra en el ejemplo, *Facebook* permite consultar el historial de notificaciones a través del perfil del usuario.

Descripción de los modelos: la Fig. 6.17 muestra los modelos asociados al patrón. La clase principal del modelo de funcionalidad es *Notification,* que describe mediante sus atributos: cuando es generada la notificación (*generatedOn*), un enlace para navegar hacia la información asociada (*linkTo*) y hasta cuál fecha la notificación se considera como reciente por el sistema (*availableUntil*). Esta clase principal se especializa en otras cuatro clases, que categorizan las actualizaciones más comunes que originan una notificación: un cambio en la información del perfil del usuario (*UserUpdate*), la actualización de un contenido (*ContentUpdate*), un evento próximo (*Event*) o un cambio producido en el contexto de la aplicación (*ApplicationUpdate*). La clase *User-*

Notification mantiene aquellas notificaciones que se encuentren asociadas a un usuario, ya sea por que son recientes (asociación *current*) o, porque pertenecen al historial (asociación *history*). El estado de cada notificación es modificado utilizando la operación *checked,* que provoca que la notificaciones pasen al historial y *discard*, que las elimina completamente.

El modelo de interacción define dos tareas de aplicación (*Application event* y *Notification creation*) que inician la creación de la notificación en el sistema. Una vez creada, se le muestran al usuario, mediante la tarea *List user notifications,* aquéllas que se encuentran disponibles. Desde ese momento, son ofrecidas las interacciones de selección, visualización y eliminación. Esta última impide la posterior visualización de la notificación utilizando el operador "[>". Los cambios en la notificación, si los hubiera, son guardados mediante la tarea *Store user notifications*.



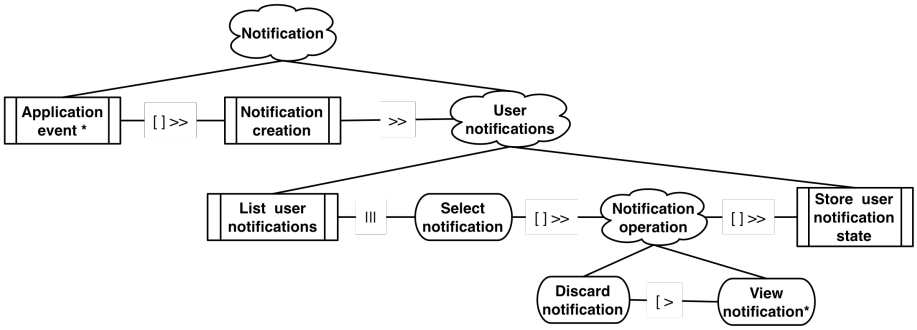


Fig. 6.17: Modelos del patrón *Notification*

6.2.3 Suggestion

Problema: proporcionar al usuario contactos y contenido que resulten de su interés, en función de sus preferencias en el marco de la aplicación.

Rationale: el patrón *Suggestion* aporta información relacionada que resulte útil al usuario de modo que este descubra otros contenidos de la aplicación. El patrón se encuentra habitualmente asociado a un contenido (por ejemplo un video) que el usuario está consultando. A partir de dicho contenido, el patrón deduce un conjunto de contenidos relacionados que poseen una temática similar. El patrón no se limita únicamente a proporcionar contenido relacionado, sino que también puede aplicarse en el ámbito de los contactos. Por ejemplo, en el marco de las aplicaciones sociales, este patrón permite que el usuario localice personas a partir de la red de vínculos sociales de sus contactos o a partir de sus lugares de trabajo o estudio. La ventaja más evidente de este patrón es la de recuperar la relación con personas con las cuales se ha perdido el contacto. Por lo tanto, este patrón ayuda a explotar las redes sociales presentes en la Web 2.0.

La principal diferencia de este patrón con el patrón *Notification*, es que se tiene en cuenta el estado actual del usuario en el marco de la aplicación, es decir, que información está recibiendo y sus contactos actuales. Mientras que las notificaciones se producen automáticamente, a la hora de generar las sugerencias, se consideran las preferencias específicas de cada usuario. En con-

secuencia, las notificaciones suelen ser compartidas entre varios usuarios mientras que las sugerencias son específicas de cada uno.

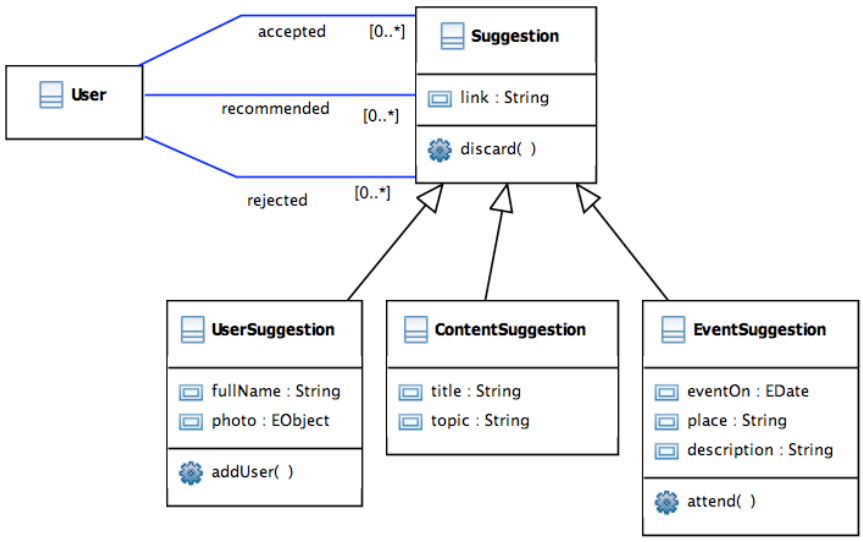
Contexto de uso: en cualquier aplicación de carácter social se recomienda el uso de este patrón puesto que facilita la localización de nuevos contactos. Muchas aplicaciones incluyen el patrón en el proceso de registro, con el objetivo que el usuario establezca un conjunto inicial de contactos en su red social. Otro escenario en el cual el patrón se aplica frecuentemente son los portales de contenido multimedia. En estos contextos se recomienda su uso simultáneo con el patrón *Tag Definition* puesto que, este último patrón, permite asociar información útil para que, posteriormente, la aplicación genere las recomendaciones.

Solución: la implementación más habitual del patrón es mostrar el conjunto de sugerencias, en el punto de la aplicación en el cual son relevantes. De esta forma, por ejemplo, cuando el usuario consulta su lista de contactos, el sistema le ofrece una lista de personas que tal vez conozca. Las recomendaciones se definen como una lista de *items* situados de modo no intrusivo en la interfaz. Cuando el usuario selecciona uno de los *items*, puede realizar las operaciones oportunas como, por ejemplo, añadir el nuevo contacto a su red social.

Ejemplo: la Fig. 6.8 muestra la implementación del patrón en las aplicaciones Web *Facebook* (izquierda) y *YouTube* (derecha). En el primer ejemplo, el patrón es utilizado para proporcionar al usuario un conjunto de personas que tal vez conozca, ya sea por que tienen contactos en común o por que comparten intereses. Estas personas son representadas mediante una fotografía incluyendo la posibilidad de invitarlas a la red personal del usuario. El segundo ejemplo muestra un conjunto de videos relacionados con el contenido que actualmente se encuentra consultando el usuario. Al igual que en el ejemplo anterior, cada video es representado mediante una captura de pantalla y un conjunto de palabras clave, que se relacionan con la temática del video reproducido. Cabe destacar que ambas implementaciones "aprenden" de la decisiones del usuario proporcionando, en un futuro, recomendaciones más acordes con sus gustos.

Descripción de los modelos: la Fig. 6.18 muestra los modelos asociados al patrón. El modelo de funcionalidad muestra la clase *Suggestion* relacionada con *User* a través de tres relaciones de asociación. Cada sugerencia que genera el sistema se relaciona con un usuario específico (relación *recommended*). Cuando el usuario acepta o descarta la sugerencia (operación *discard*), ésta pasa a relacionarse mediante las asociaciones *accepted* o *rejected* respectivamente. Al igual que las notificaciones, las sugerencias se clasifican en tres tipos en función del tipo de recomendación que se proporciona. Estas tres clases guardan en común un enlace para acceder a la información detallada de la sugerencia. La información y funcionalidad ofertada varia en función del tipo de sugerencia. En consecuencia, las recomendaciones sobre usuarios (*User-Suggestion*) muestran el nombre completo y la foto del usuario recomendado, además de incluir la operación *addUser* para agregarlo a nuestra red social.

El modelo de interacción del patrón resulta sencillo de definir. Básicamente, el sistema se encarga de generar, en función de la información que disponga, un conjunto de sugerencias (*Compute suggestions*). El usuario, por su parte, acepta o rechaza una sugerencia de tal manera que su elección es almacenada, con el propósito de ser considerada a la hora de generar un nuevo conjunto de sugerencias.



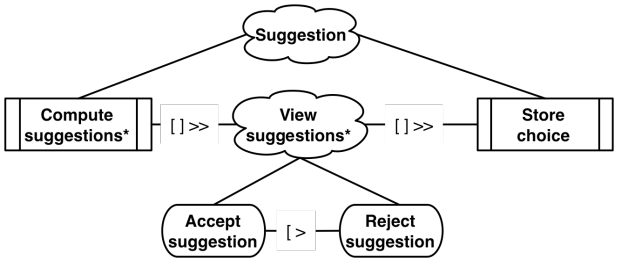


Fig. 6.18: Modelos para el patrón *Suggestion*

6.2.4 Public Profile

Problema: el usuario quiere dar a conocer su presencia en una aplicación Web, pero manteniendo la privacidad de parte de su información.

Rationale: la privacidad es una característica esencial a tener en cuenta en el ámbito de la Web y sobretodo, en las aplicaciones de carácter social. Sin embargo, este requisito es opuesto al de darse a conocer a través de la Web 2.0, puesto que es necesario definir al menos un conjunto mínimo de información que permita al resto de usuarios identificarnos. Al publicar la información en una aplicación Web social, el usuario se expone a que ésta sea fácilmente localizable, utilizando motores de búsqueda, con solo conocer su nombre. Este hecho es simultáneamente una ventaja y un inconveniente, puesto que facilita a cualquiera recabar información sobre nosotros estemos interesados o no.

El patrón *Public Profile* es una forma sencilla para evitar este inconveniente. Las aplicaciones Web que utilizan este patrón, tan solo autorizan a los motores de búsqueda, la indexación de la página Web que representa el perfil público, de tal forma que el resto de páginas de privacidad más elevada se mantienen ocultas. Este patrón también es utilizado por muchos usuarios para definir su página Web personal, de un modo sencillo, sin tener que recurrir a herramientas de desarrollo Web. En definitiva, queda patente que el objeto del patrón es generar la identidad virtual de un usuario en la Web.

Contexto de uso: este patrón es aplicado en aquellas aplicaciones en las cuales el usuario define un perfil con información, tanto de carácter personal

como privado. En consecuencia, un requisito necesario es que exista la introducción de dicha información. La gran mayoría de las aplicaciones Web 2.0 incluyen un proceso de registro y de creación de un perfil, si bien en muchas de ellas este proceso no es obligatorio. Otro escenario en el cual es aconsejable la aplicación del patrón es cuando se desea revelar detalles de la información personal, gradualmente, dependiendo del contacto. Si alguien desea recibir información más detallada, queda a decisión del usuario otorgarle acceso a un nivel de privacidad más elevado. Este patrón también es utilizado para generar, automáticamente, una "tarjeta de presentación" del usuario en el ámbito de la Web.

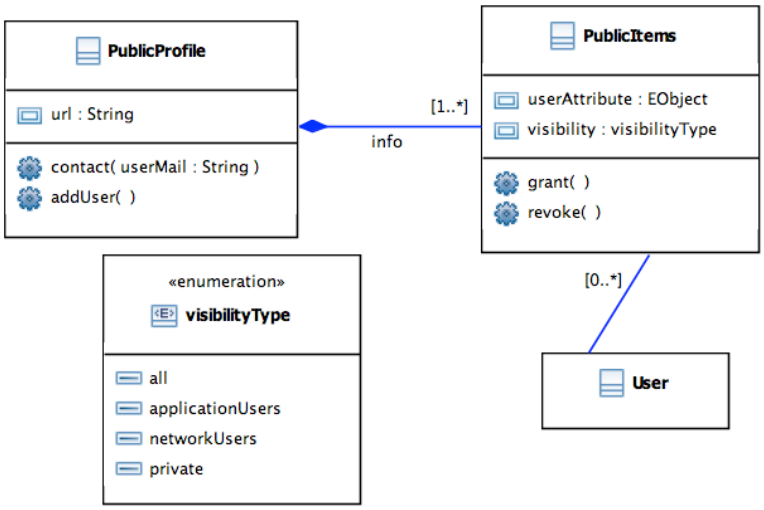
Solución: la implementación más habitual del patrón consiste en una página Web en la cual únicamente se muestra la información pública. Desde dicha página Web, se ofrece la funcionalidad para enviar algún tipo de mensaje al usuario, para ponerse en contacto con él o para agregarlo a la red de contactos. Además, esta página es accesible mediante una URL pública y exclusiva del usuario, que permite acceder al resto de su perfil, si se otorgan los permisos correspondientes.

Ejemplo: la Fig. 6.10 muestra la aplicación del patrón en el sitio *LinkedIn*. A través de la herramienta de configuración del sitio Web se especifica el perfil público. En primer lugar, se define una URL a través de la cual el perfil es accesible con el formato www.linkedin.com/in/<nombre seleccionado>. También se ofrece la selección de los distintos grupos de información que son mostrados en el perfil: datos personales básicos, fotografía, encabezado personal, resumen del currículum, especialidades, puestos de trabajo actuales y pasados, educación, sitio Web personal, intereses, grupos y premios o menciones. Cada uno de estos grupos de información son mostrados o no, a elección del usuario.

Descripción de los modelos: la Fig. 6.19 muestra los modelos que describen el patrón. La clase *PublicProfile* se compone de la URL y dos operaciones: una para enviar un correo al usuario y otra para agregarlo a la red del visitante del perfil (siempre y cuando se encuentre registrado en la aplicación). Un *PublicProfile* se compone de uno o más *PublicItems* que hacen referencia a atributos de información personal, que pueden aparecer en el perfil. Por lo tanto, el patrón realmente define una vista pública sobre la información dis-

ponible de un usuario en el sistema. La privacidad es acotada, utilizando el tipo enumerado *visibilityType* que define la visibilidad de cada ítem de información: pública para cualquier usuario (*all*), visible para aquellos usuarios registrados (*applicationUsers*), visible para usuarios registrados en la aplicación y que pertenezcan a alguna de las redes del usuario (*networkUsers*), y privada (*private*). Esta visibilidad se establece mediante las operaciones *grant* y *revoke*, disponibles en la misma clase.

Respecto al modelo de interacción, se describe la interacción necesaria para la creación del perfil. La consulta se descarta puesto que corresponde a una interacción habitual con una página Web. En el modelo, se definen dos tareas abstractas principales que sirven para la definición de la URL (tarea *Define URL*) y, la definición de la visibilidad de cada elemento de información (tarea *Set profile*). Esta última tarea, se descompone en varias subtareas porque es necesario que el sistema proporcione al usuario, qué elementos de información aparecen en el perfil (tarea *List profile item)* y que éste establezca la visibilidad de cada uno (tarea *Set item visibility*).



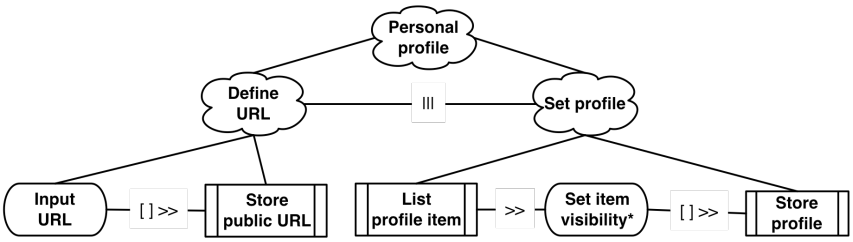


Fig. 6.19: Modelos para el patrón *Public Profile*

6.2.5 Subscription

Problema: el usuario quiere recibir actualizaciones periódicas cuando ocurre algún cambio sobre un elemento de la aplicación.

Rationale: el patrón *Subscription* guarda una estrecha relación con la sindicación basada en *feeds* RSS, tan habitual en los portales de información. Los llamados *feeds* RSS son fuentes de difusión de noticias basados en una especificación XML estándar, que facilitan la recepción de información actualizada sobre distintas temáticas. Cuando se añade alguna noticia a un *feed* en el cual se encuentra suscrito el usuario, éste recibe en una aplicación de correo electrónico o en un lector de *feeds*, una pequeña descripción de la noticia y el enlace a la información más detallada. Este mismo funcionamiento se ha trasladado al marco de las aplicaciones Web 2.0, en donde cualquier contenido o usuario es un *feed* potencial, y el lector de dichos *feeds* es la propia aplicación.

El patrón guarda dos diferencias fundamentales con respecto al patrón *Notification* que son las que lo caracterizan. En primer lugar, el patrón *Subscription* implica un interacción activa por parte del usuario que es quién decide las notificaciones a recibir. En segundo lugar, proporciona un mayor detalle de la información que el patrón *Notification*. Tomando como ejemplo un evento sobre la realización de un concierto, mediante el patrón *Notification,* el usuario recibe información sobre cuándo y dónde se produce dicho concierto, mientras que mediante el patrón *Subscription*, el usuario recibe no solo esta información, sino también los cambios que se producen en el contexto de dicho concierto (día que se ponen a la venta las entradas, horario definiti-

vo, notas de prensa, etc.). Además, mediante la aplicación del patrón, el usuario puede subscribirse a conciertos similares u a otros conciertos del mismo grupo.

Por otra parte, mediante el patrón *Subscription* se obtiene, indirectamente, el interés sobre un determinado contenido o la popularidad de los usuarios. Aquellos contenidos y usuarios que poseen un mayor número de suscriptores, se consideran más populares en el ámbito de la aplicación. De hecho, el índice de subscripciones se utiliza en varias aplicaciones Web 2.0 como un equivalente del índice de popularidad o relevancia.

Contexto de uso: este patrón se utiliza en aquellos escenarios en los cuales el usuario quiere estar informado, continuamente, sobre los cambios que se producen en el marco de la aplicación. Aunque comparte el objetivo del patrón *Notification*, puesto que ambos patrones informan sobre cambios, en el patrón *Subscription* es el usuario quien decide, y no la aplicación, qué cambios desea recibir. Un escenario de uso habitual es la selección de la temática de las noticias producidas en la aplicación que quieren recibirse. También es posible la subscripción, no solo al contenido, sino a otro usuario, práctica muy habitual en aplicaciones de *microblogging*. En este contexto de uso, cuando el usuario realiza algún cambio es automáticamente enviado a todos sus usuarios suscritos.

Solución: la solución más habitual de este patrón es la definición de un servicio de subscripción, que reciba como argumento el contenido o usuario al cual desea subscribirse. Por la tanto, la única interacción que tiene que realizar el usuario es la de ejecutar dicho servicio. Una vez suscrito, la nueva información que se va produciendo se agrega, directamente, a una página personalizada que informa de los distintos cambios. La implementación habitual de esta visualización es similar a la utilizada en el patrón *Notification*.

Ejemplo: la Fig. 6.14 muestra un ejemplo del patrón *Subscription* en el sitio Web *Twitter*. El ejemplo muestra dos grupos de noticias a los cuales puede subscribirse el usuario haciendo clic en el botón "seguir." El sitio Web *Twitter* introduce el concepto de seguidor (*follower* en inglés) para determinar que personas o entidades son más populares, en función de su número de subscripciones. Además, desde el perfil del usuario, se consulta no solo a que

contenido está suscrito, sino también que personas están suscritas o "siguen" al propio usuario.

Descripción de los modelos: la Fig. 6.20 muestra los modelos que representan el patrón. El modelo de funcionalidad define una clase *Subcription*, la cual incluye las operaciones para crear o cancelar la subscripción. En dicha clase se incluye también un resumen de la subscripción y la periodicidad, definida mediante el tipo enumerado *SubscriptionPeriodicity*, con la cual se le informa al usuario. Esta clase siempre se encuentra relacionada con un usuario que recibe la subscripción a través de la asociación *suscribedTo* y, un usuario o contenido sobre el cual se generan nuevos *SubscriptionItems* a través de la asociación *ListenerTo*. Una subscripción solo se relaciona con un usuario o con un contenido a través de esta asociación, pero no simultáneamente. Los distintos *SubscriptionItems* son recibidos y consultados por el usuario, pudiendo ser eliminados a través de la operación *discard*.

Respecto al modelo de interacción, el patrón se compone de dos tareas abstractas, en un primer nivel, que representan tanto la creación como la recepción de subscripciones. La interacción para la creación se define con una tarea que inicia la subscripción del contenido (*Subscribe to content*) y que, posteriormente, es almacenada (*Store subscription*). A continuación, la recepción de subscripciones comienza con una tarea del sistema que muestra al usuario los nuevos ítems disponibles. Sobre este listado, se realizan las interacciones para consultar el ítem o descartarlo. Estas se encuentran engobladas en la tarea abstracta *Subscription operations*.

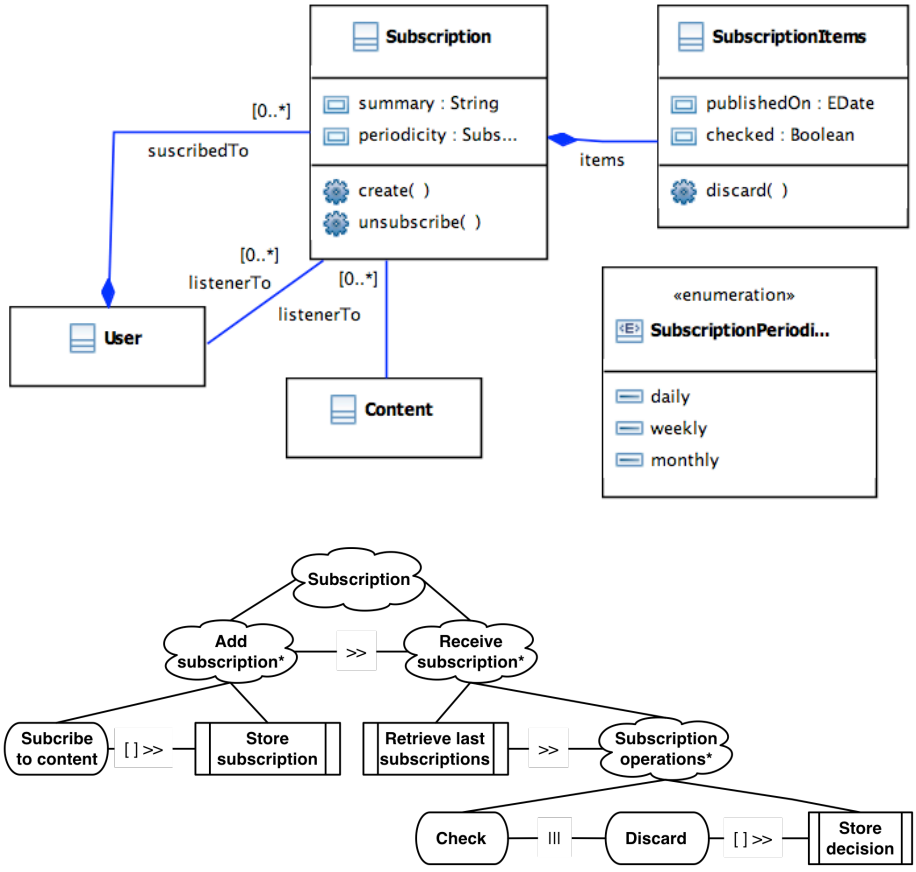


Fig. 6.20: Modelos para el patrón *Subscription*

6.3 Incorporación en un método de Ingeniería Web

En la sección anterior, se han documentado una serie de patrones que son ampliamente utilizados en el marco de las aplicaciones Web 2.0. Mediante la incorporación de modelos conceptuales, se formaliza esta documentación y se mejora la comprensión del patrón por parte del analista. Sin embargo, el objetivo de este trabajo no es solo analizar, modelar y documentar estos patrones Web 2.0, sino también proponer una estrategia para la inclusión de los mismos en un método de Ingeniería Web dirigido por modelos. En la

presente sección, se define y formaliza esta estrategia y como ilustración de la misma, se aplica en el contexto del método de Ingeniería Web OOWS 2.0.

6.3.1 Estrategia propuesta

La fase de análisis de la mayoría de los métodos DSDM, específicamente, aquéllos provenientes de la disciplina de la Ingeniería Web, se basa en un conjunto de primitivas de modelado o conceptuales. Estas primitivas conceptuales se formalizan y especifican en un metamodelo especifico de cada método. De este modo, cuando el analista instancia este metamodelo utilizando las primitivas conceptuales disponibles, se construye el modelo de análisis propuesto. Si se analizan los métodos de Ingeniería Web actuales, la expresividad que proporcionan sus modelos es suficiente en muchos casos para abordar los patrones aquí expuestos. En otras palabras, no es necesario incluir nuevas primitivas conceptuales en los métodos, sino utilizar las ya disponibles para modelar soluciones similares a las propuestas en el patrón.

El problema para incluir estos patrones no reside, por lo tanto, en la expresividad de los modelos conceptuales de los métodos de Ingeniería Web, sino en el proceso para su especificación. Generalmente, el modelado de estos patrones requiere la combinación de varias primitivas conceptuales y de varias vistas de los modelos del método. Por ejemplo, si tenemos en cuenta el patrón *Quick Comment,* debemos modelar, en primer lugar, un conjunto de clases que den soporte a la creación de los comentarios y, a continuación, una vista de presentación o interfaz para interactuar con dichas clases. Si el patrón se aplica de manera repetitiva en la aplicación o en varias aplicaciones, realizar este proceso de modelado una y otra vez resulta tedioso. Puesto que los patrones Web 2.0 propuestos en la presente tesis se utilizan con asiduidad, tal y como se ha evaluado en la sección 6.1, resulta determinante proporcionar un mecanismo para reusar los modelos que definen un patrón, con el fin de mejorar la eficiencia.

Una propuesta muy común para extender la expresividad de un método DSDM, como por ejemplo se muestra en [Koch, Pigerl *et al.* 2009], es la introducción de nuevas primitivas conceptuales con dicho objetivo. En consecuencia, la aplicación de esta aproximación en el marco de la presente tesis doctoral, implicaría la definición de catorce nuevas primitivas conceptuales o

en otros términos, una por cada patrón Web 2.0 analizado. Si bien esta aproximación resulta válida a nivel conceptual, presenta ciertos inconvenientes cuando se contempla desde el punto de vista del método. Cada nueva primitiva conceptual, implica modificar las reglas de transformación para generar los nuevos artefactos software que la representan, en un nivel de abstracción menor, por ejemplo, el código que implementa el patrón. Esta transformación no resulta trivial de especificar, porque existe un elevado *gap* semántico entre la primitiva conceptual y el código a generar, debido a que el patrón abstrae una solución compleja. Tal y como se ha recalcado anteriormente, mucha de la expresividad ya se encuentra presente en otras primitivas conceptuales de los métodos. A pesar de ello, la aplicación de esta aproximación implica que se tengan que definir nuevas reglas de transformación que incluyen semántica, que casi con total seguridad, ha sido previamente definida.

Desde nuestro punto de vista, una aproximación más adecuada es la reutilización de las primitivas conceptuales para la representación del patrón. En vez de crear una nueva regla de transformación desde la primitiva conceptual, que representa el patrón, al código se realiza una transformación a un modelo intermedio, compuesto por las primitivas conceptuales actuales del método. Las ventajas que proporciona está solución son, esencialmente, dos: (1) es posible reutilizar las reglas de generación de código que se han definido previamente y, (2) la definición de una transformación a un nivel conceptual similar (de modelo a modelo) resulta más sencilla, tanto de especificar como de mantener.

En consecuencia, en el contexto de la presente tesis doctoral, un patrón Web 2.0 es representado como un modelo conceptual perteneciente a un método de Ingeniería Web dirigido por modelos. Este modelo específico del método tiene que ser coherente con los modelos de funcionalidad e interacción que representan el patrón. Por coherente entendemos que, aunque el método utilice un modelo o notación distinta a la utilizada en la definición del patrón, el nuevo modelo propuesto tiene que ser semánticamente equivalente.

Teniendo en cuenta estas ideas, la visión general de nuestra propuesta es la siguiente: cuando el analista desea utilizar un patrón Web 2.0, crea una

instancia en el modelo utilizando la primitiva conceptual que lo representa. Sin embargo, en un segundo plano, se realiza un proceso de substitución que transforma dicha primitiva conceptual al modelo específico del patrón. Este mecanismo no es completamente original, puesto que ha sido aplicado en otros entornos como, por ejemplo, en la herramienta *Rational Architect* [Swithinbank, Chessell *et al.* 2005]. En dicha herramienta ,se proporciona un mecanismo similar para encapsular parte de un modelo como un patrón para, posteriormente, reutilizarlo en la construcción de otros modelos. Este encapsulamiento no se reduce únicamente a las primitivas conceptuales, sino que también incluye el código *Java* que se haya asociado al modelo.

Debido a que cada método se basa en sus primitivas conceptuales específicas, no es viable definir una transformación o especificación de los modelos del patrón Web 2.0, que pueda ser incluida simultáneamente en todos ellos. En otras palabras, para cada método se tiene que especificar un modelo que representa cada patrón. A pesar de este inconveniente, si que es posible definir una estrategia genérica que sirva de guía, y que simplifique el proceso de especificación e inclusión de los patrones en el método. La estrategia que se presenta a continuación, siguiendo estas pautas, se compone de cuatro etapas:

1. Extensión de la semántica del método: el primer paso de la estrategia consiste en verificar si las primitivas conceptuales de las cuales dispone el método, soportan o no la expresividad requerida por el patrón Web 2.0 a introducir. Esta comprobación es obligatoria puesto que posteriormente el patrón es representado utilizando estas mismas primitivas. El análisis debe contemplar el soporte tanto del modelo de funcionalidad como el de interacción. Respecto al primer modelo debe comprobarse si es posible definir las operaciones asociadas a la utilización del patrón. En el segundo caso, el modelo de interacción, se debe validar que el método permite la secuencia de interacciones que representan los operadores presentes en el CTT.

En el caso que se concluya que algún aspecto semántico no esta incluido, este se debe contemplar utilizando la primitiva o primitivas conceptuales correspondientes. La ventaja de esta aproximación es que la nueva semántica no se encapsula en el marco del patrón sino en el marco del metamodelo del método. De esta forma se enriquece la expresividad del método se use o no el

patrón y si la misma semántica es aplicada en otro patrón, esta no tiene que ser definida de nueva. Como se incluyen estas nuevas primitivas es un aspecto específico de cada método y por lo tanto, queda fuera del alcance de la tesis.

2. Redefinición del metamodelo del método: Una vez incluida la semántica necesaria, el metamodelo que define las primitivas conceptuales del método tiene que ser extendido. Esta extensión, a diferencia del paso anterior, no consiste en incluir nueva semántica, sino en una única primitiva conceptual que represente al patrón Web 2.0. Como mínimo, el patrón debe estar relacionado, mediante una asociación unaria de composición, con otra primitiva conceptual del metamodelo. Esta relación establece el punto de extensión metodológico a partir del cual el patrón es instanciado al crear un modelo, tal y como muestra la Fig. 6.21. En el caso mostrado, el patrón se relaciona con la primitiva conceptual C de tal manera que, previamente a la instanciación del patrón, tiene que crearse una instancia de dicha primitiva conceptual. Además, las distintas propiedades de la primitiva conceptual se utilizan como parámetros que toman valores al ser instanciados en un modelo. Estas propiedades son tanto atributos de la primitiva conceptual como relaciones con otras primitivas. Ambos, atributos y relaciones, son utilizados en las siguientes fases, como información adicional que sirva para configurar el comportamiento del patrón Web 2.0.

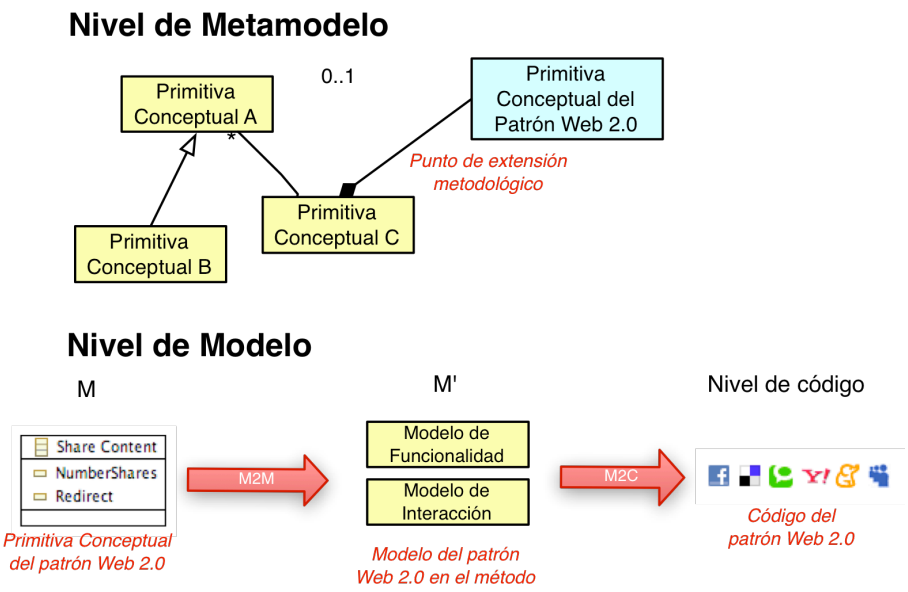


Fig. 6.21: Proceso de generación del modelo del patrón Web 2.0

3. Generación del modelo del patrón Web 2.0: Una vez que el método posee tanto la semántica como la primitiva conceptual asociada al patrón, se especifica el modelo que representa al patrón en el método. En esta fase, se define, por lo tanto, el modelo implícito que representa la primitiva conceptual del patrón. Este modelo implícito se define mediante una transformación modelo a modelo (M2M) de refinamiento: una transformación M2M en la cual los metamodelos origen y destino de la transformación son el mismo. Este es el caso que nos ocupa, ya que la primitiva conceptual (modelo origen) forma parte del mismo metamodelo, tal y como se ha indicado en el paso anterior.

El objetivo de la transformación es el de refinar o substituir esta primitiva conceptual, por un conjunto de primitivas conceptuales que representan su semántica en el marco del método. La Fig. 6.21 (debajo) resume el proceso aquí descrito. El analista crea un modelo M en el cual se usa una primitiva conceptual asociada a un patrón Web 2.0, en el ejemplo, el patrón *Share Content*. A continuación, el modelo M es transformado en un modelo M' mediante la transformación de refinamiento correspondiente. Puesto que el modelo M' es conforme al metamodelo del método, puede generarse el códi-

go correspondiente utilizando las transformaciones modelo a código previamente definidas.

Las ventajas de esta aproximación son tres: (1) el modelo específico del patrón, en el marco del método DSDM, es formalizado mediante un lenguaje de transformación de modelos; (2) al situar la introducción del patrón Web 2.0 a un nivel de abstracción mayor que él de implementación, se simplifica la complejidad de esta tarea; (3) las reglas M2C, ya presentes en el método, son reutilizadas no siendo necesario definir nuevas. Es más, si estas reglas fuesen necesarias se habrían definido, anteriormente, en el paso uno de la estrategia mediante la introducción de nuevas primitivas conceptuales.

4. Soporte a nivel de herramienta: Finalmente, la herramienta de soporte al método tiene que incluir la nueva expresividad introducida. Este soporte implica: (1) proporcionar una notación para especificar tanto el patrón Web 2.0 como sus atributos y relaciones, y (2) la implementación las transformaciones M2M necesarias. En el caso que nueva semántica haya sido introducida en el primer paso, también se tienen que soportar las nuevas primitivas conceptuales.

La estrategia aquí propuesta ha sido aplicada en el método OOWS 2.0 con el objetivo de validarla. En concreto, se han soportado los cinco patrones introducidos en la sección 6.2. Con el objeto de ilustrar esta estrategia, se muestra a continuación, en detalle, la introducción de los patrones *Quick Review* y *Notification*.

6.3.2 Aplicación en el método OOWS 2.0

Para mostrar la aplicación de la estrategia, a continuación se muestra la introducción de los patrones *Quick Review* y *Notification*. En concreto, se van ilustrar utilizando dos requisitos extraídos del ejemplo ilustrativo *my Social Research*, definido en el Anexo A.1:

- Los diferentes artículos pertenecientes a un usuario pueden ser revisados por otros usuarios. Estos proporcionan, tanto su evaluación personal ,como su opinión en forma escrita sobre el artículo. Estos

comentarios y valoraciones aparecerán como información adicional sobre el artículo.

- Cuando los usuarios entran en el sistema, deben ser informados automáticamente de fechas importantes (llamada a la participación, fecha limite de envío de contribuciones, celebración, etc.) relacionadas con las conferencias de sus áreas de interés.

El primer requisito hace referencia al uso del patrón *Quick Review*, mientras que el segundo es resuelto mediante el patrón *Notification*. En el marco del método OOWS 2.0, el modelo de funcionalidad del patrón es soportado por el Modelo de Objetos, Funcional y Dinámico de OO-Method, mientras que el modelo de interacción es soportado por el Modelo de Interacción Abstracto y el Modelo de Interfaz RIA, descritos en la presente tesis. En el presente capítulo no se aborda la implementación, a nivel de herramienta, de dicha introducción, pero si se presenta la transformación necesaria a nivel conceptual. A continuación, se pasa a detallar la introducción de ambos patrones.

6.3.2.1 Introducción del patrón Quick Comment

En primer lugar, se tiene que comprobar si los modelos actuales del método permiten la representación de la funcionalidad. El patrón *Quick Comment* plantea un modelo de funcionalidad relativamente sencillo, que se basa en una clase que almacena los comentarios, asociándolos al contenido correspondiente, y en un método para la creación de estos comentarios (ver Fig. 6.1). El patrón incluye funcionalidad adicional para responder y suscribirse, pero dicha funcionalidad tampoco resulta compleja de definir, porque se especifica mediante eventos del Modelo de Objetos. Desde el punto de vista de la interacción, el requisito más complejo es la aparición inmediata de los comentarios en la interfaz al ser introducidos. Este requisito es más tecnológico que conceptual, puesto que este comportamiento se expresa mediante un *widget* adecuado, perteneciente al Modelo de Interfaz RIA. Por lo tanto, del análisis preliminar se concluye que no se necesitan nuevas primitivas conceptuales para incorporar el patrón.

El siguiente paso en la estrategia es la definición del punto de extensión metodológico. La Fig. 6.22 muestra como se ha definido dicha extensión. En

primer lugar, la entidad *QuickComment* permite la instanciación el patrón. Se observan dos relaciones: (1) *contentId* sobre un atributo de una clase, que se reusa como identificador de los comentarios y (2) la relación *content,* que asocia el patrón con la clase que representa el contenido comentado. Asimismo, se incluyen cuatro atributos que configuran la semántica del patrón. Los dos primeros, *allowEdition* y *allowReply*, activan los mecanismos necesarios para editar y responder a los comentarios sobre el contenido. El atributo *Subscription* activa la posibilidad que el usuario reciba los cambios realizados sobre sus comentarios y, por último el atributo *anonymous,* activa la posibilidad que usuarios anónimos también realicen comentarios.

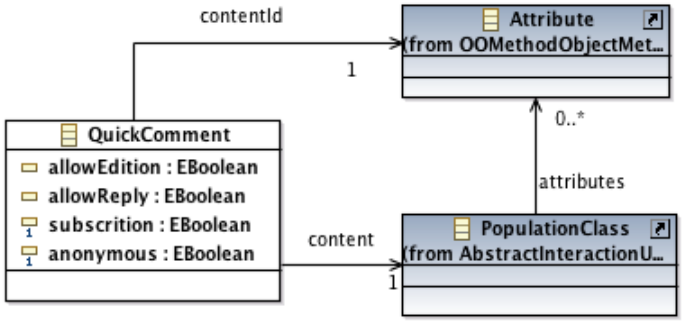


Fig. 6.22: Punto de extensión metodológico para el patrón *Quick Comment*

La Fig. 6.23 muestra el requisito como un modelo perteneciente al método OOWS 2.0 compuesto por una *Population AIU*. La AIU utiliza la clase directora *Paper* para visualizar un conjunto de artículos junto con información adicional, como su tipo o palabras clave, a través de tres clases complementarias. Para representar los patrones en las AIU de OOWS 2.0, se ha decidido utilizar como notación una clase estereotipada mediante la palabra *Pattern*. Los distintos atributos de la instancia del patrón son instanciados a través de la notación habitual para definir valores por defecto en UML. Esta representación de los patrones se ha considerado apropiada puesto que mantiene la homogeneidad con la notación actual. De este modo, la clase *pattern Quick Comment* se asocia a la clase *Paper*, indicando que los comentarios se realizan sobre los artículos listados en la AIU. El valor por defecto de los distintos atributos indican: que el identificador del contenido es el titulo

del artículo, que se habilitan la respuesta y la edición, y que no se permite ni la suscripción ni los comentarios anónimos.

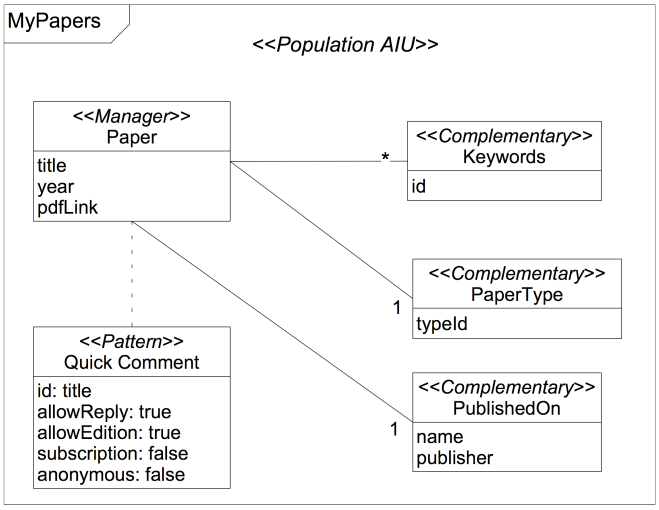
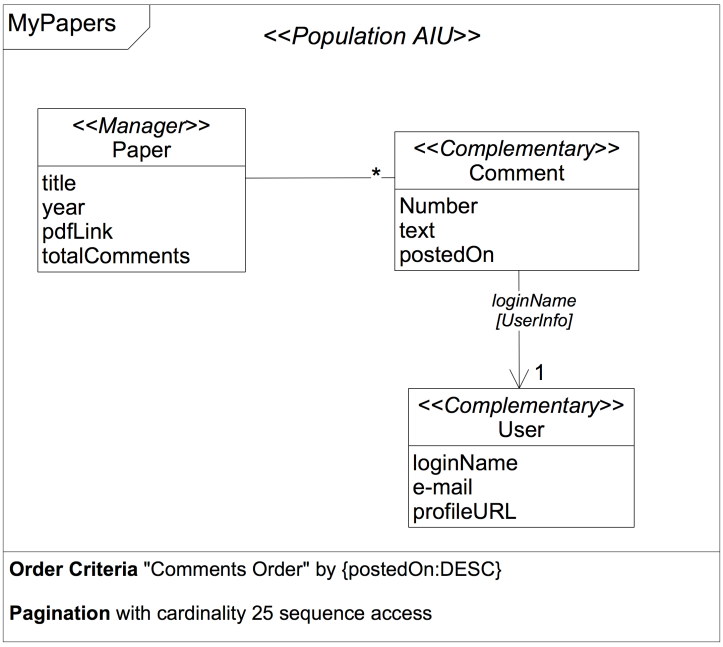


Fig. 6.23: Aplicación del patrón *Quick Comment* en OOWS 2.0

El modelo resultante de la aplicación del patrón se muestra en la Fig. 6.24. El modelo se compone de cuatro nuevas AIU: una de Población y tres de Servicio. La AIU de población sustituye a la AIU de la Fig. 6.23, incluyendo dos nuevas clases complementarias para soportar la información del patrón. Las clases complementarias que mostraban información adicional sobre el artículo no se incluyen en la figura, para favorecer la legibilidad del nuevo modelo. Básicamente, las dos clases añadidas se encargan de mostrar los comentarios y la información sobre los usuarios que los han creado. Estas clases son una representación directa de las clases propuestas en el modelo de funcionalidad, que se muestra en la Fig. 6.16. No obstante, tan solo se muestran aquellos atributos que son relevantes para mostrar la información sobre los comentarios.

La AIU incorpora, además, dos AIP para adecuarse más fielmente al comportamiento del patrón. Específicamente, se ha definido un *Order Criteria AIP*, para mostrar los comentarios más recientes, y un *Pagination AIP*, para mostrarlos en bloques de veinticinco y secuencialmente. Respecto a las *Service AIU*, cada una representa un servicio perteneciente a la clase *Com-*

ment, que es utilizado para modificar los comentarios. Cabe reseñar, que salvo la *Service AIU* para crear nuevos comentarios (*Post new comment*), las otras dos no son de generación obligatoria y son consecuencia del valor de los atributos *allowEdition* y *allowReply*, a la hora de instanciar el patrón Web 2.0.



Service AIU	Post new comment				
Operation	Comment.New				
Input Arguments	id	Alias	Type	default value	constant
	text	Comment to Paper	String	"Enter your text here"	true

Service AIU	Edit comment				
Operation	Comment.edit				
Input Arguments	id	Alias	Type	default value	constant
	commentId		Integer		true
	newComment		String		false

Service AIU	Reply to comment				
Operation	Comment.reply				
Input Arguments	id	Alias	Type	default value	constant
	commentId		Integer		true
	newComment		String		false

Fig. 6.24: Modelo resultante del patrón *Quick Comment*

6.3.2.2 Introducción del patrón Notification

Como primer paso para la introducción, se analiza el soporte a nivel conceptual por parte del método OOWS 2.0. La funcionalidad representada por el patrón *Notification* se basa en la creación de un conjunto de elementos de información, cuando se cumple una condición determinada, que se asocian automáticamente a un usuario. El método OOWS 2.0 soporta el listado de las notificaciones asociadas a un usuario, puesto que es posible modelar y obtener información del usuario conectado en la aplicación. Respecto a los eventos, éstos también son soportados, pero debe utilizarse un concepto heredado del modelo dinámico de OO-Method: los *triggers*. Un *trigger* se define mediante una clase y una condición que se basa en el estado de sus atributos. Cuando esta condición se cumple, el *trigger* lanza una operación perteneciente a alguna de las clases del sistema. Utilizando este mecanismo, se captura la semántica necesaria del patrón, por lo tanto al igual que en el caso anterior, no es necesario extender las primitivas conceptuales del método.

El punto de extensión metodológico se muestra en la Fig. 6.25. La clase *Notification* representa al patrón y se compone de dos atributos: *condition,* para definir la fórmula que genera la notificación e *historical,* que define si se tiene que almacenar el histórico de notificaciones. Esta entidad se encuentra relacionada con dos clases del Modelo de Objetos, una que representa al usuario que recibe las notificaciones (relación *notifyTo*) y, otra sobre la cual se aplica la condición para iniciar la notificación (relación *infoClass*). También se establece una relación con un *Navigation AIP* para mostrar la información asociada con la notificación. La entidad *Notification* se especializa en otras cuatro entidades, cada una de las cuales, representa un tipo de notificación tal y como se definió en la sección 6.2.2.

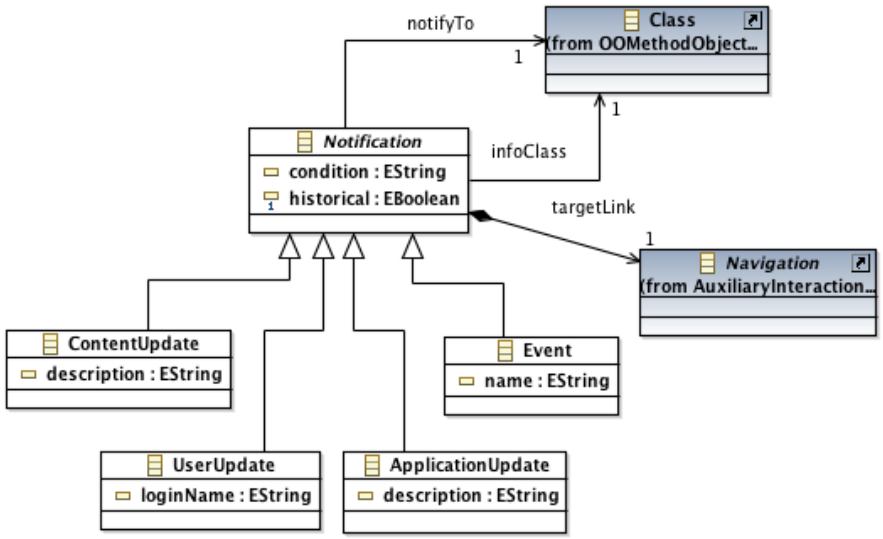


Fig. 6.25: Punto de extensión metodológico para el patrón *Notification*

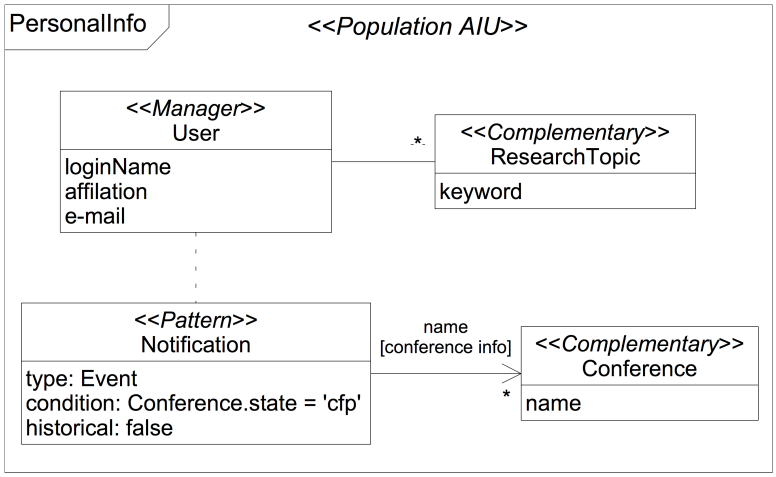


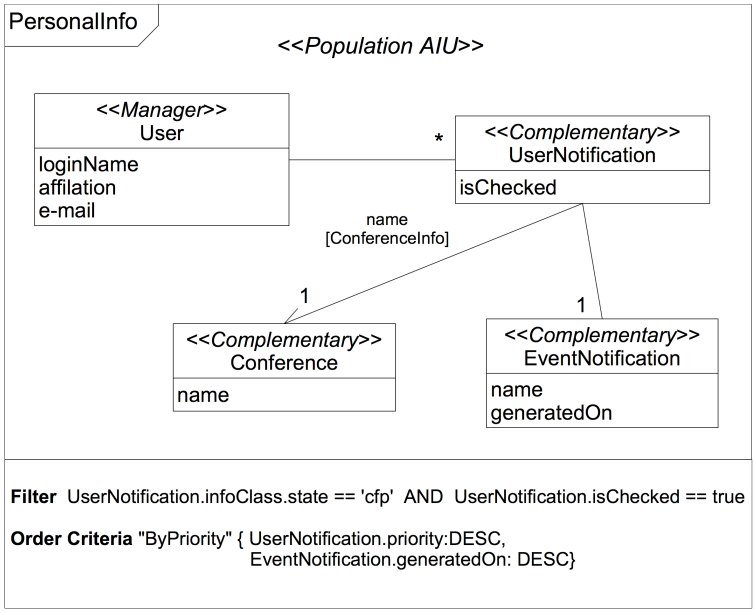
Fig. 6.26: Aplicación del patrón *Notification* en OOWS

La Fig. 6.26 muestra un ejemplo del uso del patrón, con el fin de informar al usuario sobre las nuevas conferencias dadas de alta en la aplicación. El contexto representado, *PersonalInfo*, muestra un conjunto breve de información sobre el usuario y sus áreas de investigación de interés. Para modelar la notificación, la entidad que representa el patrón se asocia a la clase *User*. Esta clase especifica el usuario receptor de las notificaciones, mediante una línea discontinua que se corresponde con la extensión metodológica del patrón. Además, esta entidad se asocia con la clase *Conference*, utilizando un *Object Navigation AIP*. Esta relación navegacional es fundamental porque indica: (1) sobre qué información se van a definir las notificaciones y (2) el Contexto de Interacción que muestra la información detallada de la notificación.

A continuación, se establece la condición que genera las notificaciones. Esta condición utiliza el atributo *state* de la clase *Conference,* para generar una notificación siempre que una conferencia se encuentre en el estado de "llamada a la participación" (*Call For Papers*). En consecuencia, se informa al usuario de las conferencias a las cuales puede enviar trabajos. Como atributos opcionales de configuración, se indica que la notificación es del tipo *Event* y que no se desea mantener un histórico con las notificaciones. Aunque en el requisito no se plantea, se puede mejorar el modelo, mediante un *Filter AIP* sobre la clase *Conference*, para notificar únicamente las conferencias de una temática concreta.

La Fig. 6.27 muestra el modelo OOWS 2.0 resultante de la aplicación del patrón. En este caso, se incluyen tres nuevas clases, *UserNotification, Conference* y *EventNotification* que se encargan de mostrar las notificaciones asociadas al usuario y que, todavía, no han sido descartadas. La AIU incluye dos AIP para complementar la semántica del patrón. En primer lugar, un *Filter AIP* que tan solo muestra aquellas notificaciones que no han sido consultadas (*isChecked* igual a falso) y, cuyo estado todavía es igual a "cfp". Como se puede observar en la fórmula, el primer átomo de la condición de filtrado se establece a partir de la información del patrón, mientras que el segundo átomo es constante para todos los casos. En segundo lugar, la AIU incluye un *Order Criteria AIP* que muestra las notificaciones, primero, por el orden de prioridad y, a continuación, por el orden de generación.

El modelo también incluye dos *Service AIU* tal y como muestra la Fig. 6.27. Estas AIU se encargan de ejecutar la funcionalidad asociada a la eliminación de una notificación (*Discard notification*) y, a la comprobación o consulta de una notificación (*Check notification*). Su especificación resulta sencilla, dado que ambas operaciones únicamente reciben el identificador de la notificación.



Service AIU	Discard notification				
Operation	UserNotification.discard				
Input Arguments	id	Alias	Type	default value	constant
	notificationId		Integer		true

Service AIU	Check notification				
Operation	UserNotification.checked				
Input Arguments	id	Alias	Type	default value	constant
	notificationId		Integer		true

Fig. 6.27: Modelo resultante del patrón *Notification*

6.3.2.3 Especificación de las transformaciones de los patrones

Un objetivo de la presente tesis doctoral es el de integrar los patrones Web 2.0 en el marco de OOWS 2.0. Para realizar esta integración, se ha decidido por la utilización de transformaciones modelo a modelo que representen, formalmente, el patrón a introducir. Por lo tanto, es necesario implementar dichas transformaciones en un lenguaje M2M adecuado como QVT [OMG 2008] o ATL [INRIA 2007]. En la presente sección, se describe como se ha llevado a cabo esta implementación en el marco del método OOWS 2.0. El objetivo de la presente tesis no es el de implementar, con detalle, todo el conjunto de reglas de transformación que especifican los patrones Web 2.0. Sin embargo, en la presente sección se esboza el procedimiento general a seguir. Para ilustrar este procedimiento se ha seleccionado el patrón *Quick Comment* como ejemplo del modelo a generar, y como lenguaje de transformación M2M se ha escogido ATL. A continuación se presenta brevemente este lenguaje.

ATL es un lenguaje de transformación entre modelos que sigue los paradigmas de programación imperativo y declarativo. Una transformación en ATL se compone de un conjunto de reglas, que definen como los elementos de un modelo origen son transformados a un modelo destino. Dependiendo del paradigma utilizado se distinguen dos tipos posibles de reglas: *Matched*

rules (declarativo) o *Called rules* (imperativo). En el marco de la presente tesis, se utilizan fundamentalmente el primer tipo de reglas. Las *Matched rules* se definen en función de una primitiva conceptual que inicia la ejecución de la regla. Esta primitiva conceptual se especifica mediante la cláusula *from*, mientras que mediante la cláusula *to,* se especifica el conjunto de primitivas conceptuales que se crean. Estas primitivas conceptuales se consideran, a partir de ese momento, como variables en el contexto de la regla pudiendo ser utilizadas en otros puntos de la transformación.

La estrategia a seguir, para introducir un patrón Web 2.0 en el método OOWS 2.0, es la definición de una regla que tome como valor de la cláusula *from,* la primitiva conceptual que representa al patrón. Como ya se trató en el punto 6.3.1, todo patrón tiene obligatoriamente una de estas primitivas definidas en el metamodelo del método. A continuación, en el cuerpo de la regla, se implementa la sustitución de dicha primitiva conceptual por el modelo OOWS 2.0 que representa al patrón. Esta sustitución abarca la creación de varios conjuntos de primitivas conceptuales pertenecientes a los modelos del método: un conjunto de clases y atributos que representan la información del patrón, las distintas AIU y, opcionalmente, un conjunto de AIP que refinan su semántica, como puede ser la inclusión de navegación. El siguiente código ATL, muestra la generación de parte del modelo de funcionalidad propuesto para el patrón *Quick Comment:*

```
rule QuickComment2OOWS {
   from
           p: OOWS!QuickComment
   to

       commentClass: OOWS!Class(
               name <- 'Comment',
               alias <- 'Comment'
               ),

       comment_at1: OOWS!Attribute(
               name <- 'Number',
               alias <- 'Number of comment',
               type <- 'Int'
           ),

       compComment: OOWS!PopulationClass(
               class <- commentClass,
               attributes <- Set{comment_at1,comment_at2,comment_at3}
       ),
```

Cada nueva primitiva conceptual se define mediante una variable precedida por dos puntos y, a continuación, la entidad del metamodelo correspondiente que la representa como, por ejemplo, *OOWS!Class.* El código mostrado genera la clase *Comment* y un atributo de la misma, "*Number of comment*", inicializando los distintos atributos de la primitiva conceptual mediante el operador "<-". Se han omitido el resto de clases y atributos, que se crean de un modo similar, para favorecer la legibilidad. Ya que, según el patrón, esta clase pertenece a una AIU de población, se define una nueva *Population Class* que hace referencia a estas nuevas primitivas conceptuales. El operador *Set* de ATL define un conjunto de elementos del modelo que se asocian, en este caso, a la relación *attributes* de la *Population Class.* El siguiente paso es asociar esta nueva clase a la *Population AIU*. Si revisamos la Fig. 6.24, se observa que la clase *Comment* juega el rol de una *Complementray Class* y que se asocia, con la clase *Content* que representa el contenido a comentar. Para establecer esta asociación se definen las siguientes primitivas conceptuales en la regla ATL:

```
commentEnd: OOWS!AssociationEnd (
        minCardinality <- 0,
        maxCardinality <- -1,
        role <- 'contentComments',
        associatedToClass <- commentClass
    ),

contentRel: OOWS!PopulationRelationship(
        source <- p.content,
        target <- compComment,
        relationshipEnd <- commentEnd
    ),
```

En primer lugar, se define un nuevo *Association End,* que define la cardinalidad en la asociación de la clase *Comment* a través de la relación *associatedToClass*. A continuación, se define una nueva *Population Relationship*, en el ámbito de la *Population AIU*, que asocia esta clase *Comment* con la clase que representa al contenido, mediante las relaciones *source* y *target.* Es interesante observar, que se ha utilizado la propiedad *content* de la primitiva conceptual que representa el patrón (*p*), como valor de la propiedad *source*. Si observamos la Fig. 6.22, esta propiedad *content* hace referencia a la clase sobre la cual se aplica el patrón *Quick Comment.*

Una vez definido el conjunto de clases del patrón *Quick Comment*, se crea la semántica auxiliar asociada al patrón. En este caso, se trata de un *Object Navigation AIP*, un *Order Criteria AIP* y un *Pagination AIP*. El código siguiente muestra la creación de estos tres mecanismos auxiliares:

```
userInfoContext: OOWS!InteractionContext(
        name <- 'userInfo',
        alias <- 'Information about users',
        specifiedByAIU <- userInfo_AIU
),

userNavigation: OOWS!ObjectNavigationAIP(
        name <- 'navToUser',
        alias <- 'navigation to user info',
        target <- userInfoContext,
        navigationAttribute <- user_at1,
        navigationalRelationship <- userRel
    ),

commentOrder: OOWS!OrderCriteriaAIP(
        name <- 'Comments Order',
        ascendent <- false,
        orderAttribute <- comment_at2
    ),

commentsPagination: OOWS!PaginationAIP(
        name <- 'Comments Pagination',
        cardinality <- 25,
        sequenceAccess <- true,
        randomAccess <- false
    )
```

En primer lugar, para definir el *Object Navigation AIP,* se especifica el *Interaction Context* destino donde se visualiza la información sobre el usuario que ha realizado el comentario. La especificación completa de este contexto se ha omitido, a fin de mejorar la legibilidad de la transformación. A continuación, se crea el propio *Object Navigation AIP*. En dicha primitiva conceptual se asocia a la propiedad *target* el nuevo *Interaction Context* y, como *navigationAttribute y NavigationalRelationship,* se asignan un atributo y una relación correspondientes a la clase complementaria *User*. En segundo lugar, la transformación muestra la creación de los AIP de *Order Criteria* y *Pagination*. En este caso, el procedimiento de generación de las primitivas conceptuales es sencillo, ya que tan solo se trata de instanciar correctamente las distintas propiedades.

El último conjunto de primitivas conceptuales a generar es el conjunto de *Service AIU* asociadas al patrón. En este caso, es obligatoria la generación de la *Service AIU* para crear nuevos comentarios, pero que, en función de los valores de la primitiva conceptual del patrón, deberán generarse o no las *Service AIU* para editar, responder o subscribirse a los comentarios. El siguiente código muestra la generación de la *Service AIU* asociada a la edición de comentarios:

```
rule QuickComment2Edition{
        from
            p:OOWS!QuickComment (
                p.allowEdition)
        to
            siuEdition: OOWS!ServiceAIU(
                inputArguments <- Set {commentId_arg,newComment_arg},
                overSystemService <- commentEditionService
            ),

            commentId_arg: OOWS!Argument(
                name <-'commentId',
                alias <- 'comment Identifier',
                type <- 'Integer'
            ),

            commentEditionService: OOWS!Service (
                name <- 'EditionService',
                alias <- 'Comment edition'
            ),
```

En esta situación particular, se define una regla específica para la generación de las primitivas conceptuales. La causa es que la generación está condicionada a los valores específicos de un atributo del patrón. Para definir esta condición se utiliza la cláusula *from*, indicando que únicamente se ejecute la regla cuando el valor del atributo *allowEdition* sea cierto. A continuación, se especifica la primitiva conceptual que genera la nueva *Service AIU*. Esta primitiva se asocia tanto a un conjunto de argumentos como a un servicio de la lógica de negocio, primitivas conceptuales que son generadas seguidamente en el ámbito de esta regla condicional.

Repitiendo el procedimiento aquí descrito, la implementación de las transformaciones del resto de patrones se realiza de manera sistemática. Aunque no se ha tratado en el ejemplo ilustrativo, es posible también la definición de *helpers* en ATL que permiten reutilizar partes de la distintas reglas

de transformación. Utilizando este mecanismo del lenguaje, se consigue simplificar y mejorar el mantenimiento de las reglas de transformación.

Además de la generación de las primitivas conceptuales, también debe contemplarse como se conservan en el modelo destino, las primitivas conceptuales del modelo original que no participan en la transformación. El procedimiento habitual en un proceso de transformación de modelos es que un modelo conforme a un metamodelo A, se transforme a un modelo que sea conforme a un metamodelo B. Tal y como se introdujo en el punto 6.3.1, en la presente tesis doctoral no se sigue este procedimiento habitual, sino que se utiliza el mismo metamodelo como origen y destino. Este modo de transformación se define como refinamiento (*refining*) y se encuentra soportado por el lenguaje ATL. Mediante este modo, las primitivas conceptuales que no son alteradas en el modelo origen se copian directamente en el modelo destino. No obstante, se tiene que cumplir una de las siguientes condiciones para que se realice dicha copia: que la primitiva conceptual participe en una regla de transformación o, que esté referenciada implícitamente por una primitiva conceptual del modelo origen. El funcionamiento de ATL, en modo de refinamiento, implica que cada vez que una primitiva conceptual del modelo destino es generada, si ésta hace referencia a otra primitiva conceptual se crea también en el modelo destino. A continuación, este proceso se repite de forma recursiva con la nueva primitiva conceptual añadida hasta que no existen nuevas referencias.

A efectos prácticos, el funcionamiento del modo de refinamiento en ATL implica la introducción de un conjunto de reglas auxiliares, con el propósito de referenciar al elemento raíz del modelo a partir del cual se definen el resto de primitivas conceptuales. En particular, teniendo en cuenta el metamodelo de OOWS 2.0, existen dos primitivas conceptuales que cumplen este rol: *Object Model* y *Abstract Interaction Model*. El siguiente código muestra las reglas necesarias para aplicar, adecuadamente, el modo refinamiento en este método:

```
module quickComment;
create OUT : OOWS refining IN : OOWS;

rule GenerateAIM {
   from
       input: OOWS!AbstractInteractionModel
   to
       out: OOWS!AbstractInteractionModel
}

rule GenerateOOM {
   from
       input: OOWS!ObjectModel
   to
       out: OOWS!ObjectModel
}
```

Observamos que la transformación del patrón *Quick Comment* se define como refinamiento mediante la palabra clave *Refining* y, que el metamodelo origen (*IN*) y el metamodelo destino (*OUT*) son el mismo. Seguidamente, se definen dos reglas, denominadas *GenerateAIM* y *GenerateOOM,* cuya función es la de copiar la primitiva conceptual raíz que representa ambos modelos de OOWS 2.0. La aplicación de estas dos reglas genera el conjunto de *Interaction Contexts, Abstract Interaction Units* y el *Object Model,* que se encuentran relacionados con estas primitivas raíz. De esta manera se garantiza que, a la hora de aplicar las transformaciones, no se pierdan las primitivas conceptuales no utilizadas del modelo original y que, además, se incluyan las nuevas primitivas definidas por el patrón.

6.4 Conclusiones

En este capítulo se ha introducido el concepto de patrón Web 2.0 y se ha ilustrado su utilidad, en el marco de los métodos de Ingeniería Web. En concreto, se han obtenido las siguientes conclusiones:

- A través del análisis de un conjunto relevante de aplicaciones Web 2.0, se ha puesto de manifiesto la amplia difusión de este tipo de patrones. Este análisis ha permitido, asimismo, determinar cuáles resultan más útiles a la hora de desarrollar una aplicación Web 2.0.

- La formalización del concepto de patrón Web 2.0 presentada incide en el uso de modelos conceptuales, aspecto fundamental, para su posterior integración en los métodos de Ingeniería Web. Al mismo tiempo, el uso de modelos ha resultado útil para comprender mejor el funcionamiento y semántica de dichos patrones.

- Se ha planteado una estrategia de integración, a nivel conceptual, que es compatible con diversos métodos de Ingeniería Web. Esta estrategia resulta novedosa, al hacer hincapié en el uso de transformaciones M2M con dicho fin.

- Como ejemplo ilustrativo de la propuesta, se ha introducido el concepto de patrón Web 2.0 en el marco del método OOWS 2.0. Mediante este ejemplo se ha dado una visión práctica de la implementación de la estrategia de integración y del proceso de transformación.

No cabe duda que existen diversos puntos a tratar con mayor detalle, como son la inclusión de nuevos patrones o la definición de una implementación más detallada de la integración. En este sentido, esta tesis se marca como meta justificar la relevancia de estos patrones en el desarrollo de aplicaciones Web 2.0 y, la necesidad de ser contemplados, a nivel conceptual en los métodos de Ingeniería Web. Como línea de investigación futura, se debe validar la integración de esta propuesta con otros métodos de Ingeniería Web.

Capítulo 7

Evaluación de la Viabilidad

El objetivo del presente capítulo es el planteamiento de una evaluación de la viabilidad de las contribuciones presentadas en el marco del método OOWS 2.0, es decir, justificar su utilidad antes de abordar su proceso de implementación. En primer lugar, en la sección 7.1, se introduce el porqué plantear una evaluación de la viabilidad. En la sección 7.2, a partir de un conjunto de escenarios extraídos de una aplicación Web 2.0 real, se ha analizado el nivel de soporte que ofrece el método OOWS original para modelar tanto la interacción como la interfaz. A continuación, se ha evaluado la mejora, en ambos aspectos, aplicando las contribuciones a nivel de modelado introducidas en la presente tesis doctoral. En tercer lugar, en la sección 7.3, se ha detallado el proceso de modelado, a nivel de interacción e interfaz, aplicando el método OOWS 2.0 sobre dos escenarios extraídos de una aplicación Web 2.0. En este proceso de modelado, se ha contemplado la inclusión de los patrones Web 2.0 presentados. Con la perspectiva de una evaluación futura con mayor rigor, en la sección 7.4 se presenta el diseño de un experimento para evaluar la mejora proporcionada por las propuestas presentadas a nivel metodológico. Las conclusiones de esta evaluación de viabilidad se presentan, finalmente, en la sección 7.5.

7.1 Introducción

Validar completamente un método de Ingeniería Web, a nivel industrial, requiere de un número elevado de recursos tales como personal, presupuesto, casos de estudio reales, realización de experimentos, etc. En el contexto de esta tesis doctoral, requeriría la implementación de toda la infraestructura tecnológica para soportar tanto el aspecto metodológico de OOWS 2.0, como la posterior producción de aplicaciones Web 2.0.

Debido a que la definición de dicha infraestructura se aleja de los objetivos de la presente tesis doctoral, se ha optado por plantear una evaluación de la viabilidad que pueda proporcionar una visión preliminar del alcance de las contribuciones. Esta evaluación de la viabilidad es un paso previo necesario ya que, si se detectan carencias en el método, pueden corregirse en una etapa temprana de su desarrollo, evitando cambios más costosos cuando éste se encuentre en las fases finales de su implantación. Asumiendo las limitaciones de los resultados obtenidos, este tipo de evaluación resulta útil para determinar si la propuesta tiene el suficiente rigor y justificar, convenientemente, el esfuerzo requerido para el desarrollo del soporte tecnológico necesario

Las contribuciones del método OOWS 2.0 han sido introducidas a nivel conceptual, es decir, se han definido nuevos modelos que abarcan la expresividad requerida por los nuevos requisitos de las aplicaciones Web 2.0. Por lo tanto, la evaluación de viabilidad que se trata en el presente capítulo tiene dos objetivos claros: (1) comprobar mediante un ejemplo real si dichos modelos son lo suficientemente expresivos para especificar una aplicación Web 2.0 y, (2) comprobar si el uso de estos modelos, realmente, suponen una ventaja al analista cuando aplica el método OOWS 2.0.

Para evaluar la propuesta se ha decidido utilizar una demostración de laboratorio o *lab demo*, tal y como se define en [Wieringa 2008], o también denominada *Synthetic environment experiment*, según la clasificación propuesta por [Zelkowitz & Wallace 1997]. Este tipo de evaluación, se basa en aplicar la propuesta sobre un ejemplo extraído de la realidad pero en un entorno artificial y controlado, como puede ser el marco de la investigación académica que nos ocupa. Una demostración de laboratorio es llevada a cabo, por el propio autor, a partir de un conjunto de escenarios seleccionados, que sean lo

suficientemente representativos, para evaluar la propuesta. Estos escenarios son presentados a través de un conjunto de tareas a realizar que son resueltas, ya sea por el mismo autor o por otros investigadores conocedores de la propuesta, aplicando las contribuciones presentadas.

El principal inconveniente de este tipo de evaluación de viabilidad es que la validez no está asegurada, porque la propuesta no es aplicada en un entorno real por personas externas a la misma. En consecuencia, el inconveniente del uso de una demostración de laboratorio, tal y como indican [Zelkowitz & Wallace 1997], es que no tiene por qué existir una correlación entre los resultados obtenidos en el entorno artificial y aquéllos que se obtendrían en un entorno industrial. No obstante, este tipo de evaluación sí que resulta útil para determinar la viabilidad de las contribuciones presentadas y, mejorarlas antes de proseguir con las etapas posteriores de aplicación.

7.2 Análisis del soporte proporcionado a nivel de modelado

Para realizar el análisis, en primer lugar se realiza una descripción de la aplicación objeto de estudio. Como ejemplo real para construir la demostración de laboratorio, se ha seleccionado la aplicación Web *23andMe* (*www.23andme.com*). Esta aplicación Web es pionera en el dominio de la información genética personalizada. La aplicación se estructura alrededor de un usuario y sus familiares para determinar, en función de su genoma, aspectos relevantes relacionados con su salud, su apariencia física y su tendencia a padecer enfermedades. El dominio de esta aplicación Web 2.0 es el de las aplicaciones de salud personalizada, también llamadas e-*Health*. Esta aplicación Web consiste, fundamentalmente, en un portal de conocimiento sobre el impacto de la información genética, en distintos aspectos fisiológicos del ser humano. A partir de una muestra de ADN enviada por correo, dicho portal genera un perfil genético personalizado que detalla la posibilidad de sufrir una determinada enfermedad o, la relación de parentesco de características genéticas relevantes. Su principal elemento diferenciador es que proporciona dicha información genética en un formato ameno y comprensible para cualquier usuario. Además, simplifica el acceso a este conocimiento genético proporcionando una interfaz sencilla y con un elevado nivel de usabilidad.

La selección de esta aplicación como demostración de laboratorio obedece a las siguientes razones:

- La Web 2.0 se ha relacionado asiduamente con aplicaciones de carácter social o de creación de comunidades. Por esta razón, es habitual realizar la asociación directa del concepto de aplicación Web 2.0 con aplicaciones tales como *Facebook, Twitter* y similares. Sin embargo, la Web 2.0 no tiene por qué limitarse únicamente a las llamadas Web sociales. El mismo principio de enfatizar y potenciar la implicación del usuario es completamente válido en otros dominios. La selección de una aplicación Web perteneciente al dominio bioinformático justifica esta afirmación al evaluar la propuesta en un ámbito, en cierta manera, alejado de las aplicaciones Web 2.0 más comunes.

- El objetivo fundamental de *23andMe* es el de proporcionar información genética de forma clara y amena al usuario. Con el fin de conseguir este objetivo, la aplicación se basa en una interfaz de usuario muy usable, visualmente agradable y altamente interactiva. Para lograr tales características, se han utilizado tecnologías RIA en su implementación. De hecho, la aplicación utiliza una serie de componentes de interfaz que no pueden ser desarrollados fuera de este ámbito tecnológico. En consecuencia, esta aplicación es una candidata ideal para evaluar el Modelo de Interfaz RIA propuesto.

- La complejidad de la aplicación, tanto a nivel de riqueza de la interfaz y como de la funcionalidad ofertada, justifican la aplicación de un método de Ingeniería Web para su especificación.

En la presente tesis, la demostración de laboratorio se centra en los escenarios que percibe un usuario registrado. Dado que el uso completo de la aplicación requiere el pago de un análisis de ADN, se ha utilizado la versión de demostración gratuita la cual utiliza un usuario con información genética ficticia. En esta versión, la funcionalidad es la misma salvo dos secciones, *Genome Sharing* y *Relative Finder*, que solo están disponibles en la versión completa y que, por lo tanto, se han omitido en el análisis.

Los distintos escenarios que ofrece la aplicación son fácilmente clasificables a partir del menú de navegación principal. Destacan cuatro secciones claramente diferenciadas: (1) *My Health*, que agrupa la funcionalidad personalizada sobre posibles enfermedades genéticas; (2) *My Ancestry*, que se compone de un conjunto de herramientas para descubrir las relaciones genéticas entre el usuario y sus familiares; (3) *Sharing & Community*, que proporciona funcionalidad para intercambiar opiniones con otros usuarios y realizar comparaciones genéticas; (4) *23andWe*, que proporciona utilidades para llevar a cabo investigaciones genéticas a partir de la información aportada por los usuarios.

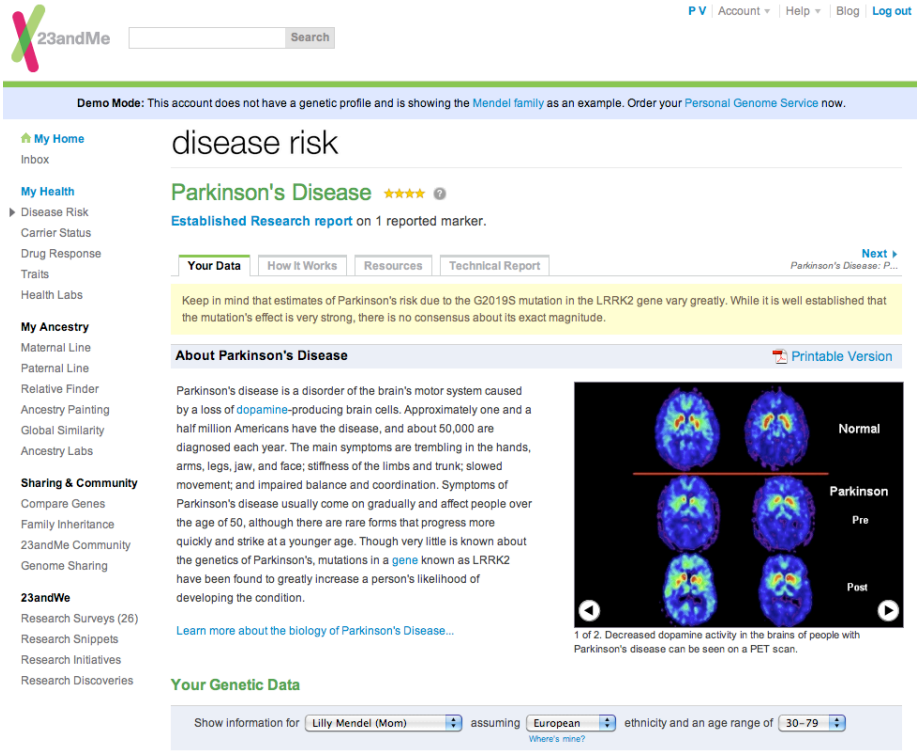


Fig. 7.1: Visión General de *23andMe*

Cada una de estas cuatro secciones se compone de varios escenarios que proporcionan una funcionalidad e interacción específicas. El sitio Web esta estructurado en pocos niveles de tal forma que en un escenario, generalmen-

te, no hay más que un nivel adicional de navegación para mostrar la información con mayor grado de detalle. A modo de especificación informal de requisitos, la funcionalidad que ofrece *23andMe* se resume en el siguiente conjunto de 18 escenarios:

1. *My Home:* muestra los mensajes de bienvenida de la aplicación junto a las últimas noticias que se hayan producido en la misma.

2. *Inbox*: define un pequeño gestor de correo para enviar mensajes privados en el ámbito de la aplicación. Permite tanto el listado de los mensajes recibidos como la redacción de los mismos.

3. *My Health*: ofrece una vista resumida de la información más reciente de los escenarios *Disease Risk, Carrier Status, Drug Response* y *Traits*. También se ofrece la navegación a la información detallada de cada uno de estos escenarios.

4. *Disease Risk*: muestra un listado de las enfermedades que, potencialmente, puede padecer el usuario en función de la información genética. Las enfermedades son clasificadas según el riesgo asociado (elevado, bajo o normal) a padecerlas.

5. *Carrier Status*: muestra un conjunto de enfermedades genéticas habituales indicando las investigaciones asociadas a cada una de ellas. Teniendo en cuenta dichas investigaciones, se indica si el usuario posee o no la variante genética susceptible de causarlas.

6. *Drug Response*: muestra la respuesta fisiológica del usuario al consumo de una droga, como puede ser el alcohol o determinados medicamentos. Para cada droga, muestra el posible efecto sobre el usuario considerando su información genética.

7. *Traits*: muestra un listado de las características fenotípicas (rasgos físicos perceptibles) del usuario que son consecuencia contrastada de su información genética. Para cada característica, se detalla la investigación que aporta validez a los datos y el grado de probabilidad que el usuario la posea.

8. *Maternal/Paternal Line:* ambos escenarios muestran el haplogrupo, materno y paterno, al cual pertenece el usuario. Un haplogrupo se define como el grupo de individuos que comparten con otro individuo, una misma variación genética en una región y tiempo determinados. Además, se explica detalladamente el origen geográfico del haplogrupo, su historia, sus relaciones jerárquicas con otros haplogrupos y la comparación con el haplogrupo de otros individuos.

9. *Ancestry Painting*: para cada cromosoma del usuario, muestra un gráfico coloreado de varios segmentos cromosómicos en función de la región geográfica de procedencia. Permite comparar esta información con la de otros familiares y grupos étnicos.

10. *Global Similarity*: muestra la similitud de la información genética del usuario con grupos de personas de otras regiones geográficas. La información se muestra, gráficamente, mediante un mapamundi con los diferentes porcentajes de similitud.

11. *Compare Genes*: compara el genoma del usuario con el de otro usuario, a fin de obtener el porcentaje de similitud genética entre ambos. La comparación puede realizarse sobre todo el genoma o, sobre genes asociados a una característica o enfermedad específica. También puede realizarse la comparación, simultáneamente, con un grupo de usuarios.

12. *Family Inheritance*: proporciona tres cálculos entre el genoma del usuario y sus familiares, para averiguar de dónde proviene el parentesco de ciertas características genéticas. En concreto, es posible calcular una comparativa entre dos familiares, el árbol genealógico de cierta característica genética y la probabilidad que los descendientes compartan algún rasgo genético.

13. *23andMe Community*: esta sección implementa un foro de participación en el cual los distintos usuarios plantean preguntas y, realizan contribuciones sobre las investigaciones que se desarrollan en la aplicación.

14. *Research Surveys*: fomenta la realización de encuestas en el marco de diversas investigaciones genéticas.

15. *Research Snippets*: plantea preguntas de respuesta rápida a los usuarios, que sirven de material para las diversas investigaciones del sitio Web.

16. *Research Initiatives*: plantea un conjunto de estudios sobre una enfermedad genética, en los cuales el usuario puede participar aportando su información personal. Cada estudio se estructura a partir de un conjunto de encuestas, que son realizadas por los usuarios, y una comunidad de participantes, con los cuales es posible interactuar.

17. *Research discoveries*: muestra los resultados de los estudios realizados gracias a la colaboración de los usuarios.

18. *Specific Reports*: este escenario se encuentra asociado a otros escenarios de la aplicación. Se encarga de mostrar la información detallada sobre una característica genética, enfermedad, investigación, encuesta u otro elemento de conocimiento de la aplicación. Consta de una vista de información, dividida, en varias pestañas que permite al usuario consultar la información de un modo estructurado. Específicamente, se muestra la explicación detallada del elemento de conocimiento genético consultado, las publicaciones que le dan soporte, el grado de influencia ambiental (causas externas que influyen en la información tratada) y, la variante genética que posee el usuario junto con el parentesco.

Para cada uno de los escenarios antes descritos, se ha analizado cual es el soporte que ofrece OOWS para modelar la interacción e interfaz necesarias. Simultáneamente, se han analizado ambos aspectos considerando las extensiones conceptuales introducidas en OOWS 2.0. Para cada escenario, se ha detallado si el aspecto está soportado completamente, es decir, si es factible modelar una interacción idéntica a la de la aplicación, parcialmente, si es posible modelar algunos aspectos pero se detectan carencias, o no soportado, si el método no tiene la expresividad conceptual necesaria para modelar el escenario. Los resultados obtenidos del análisis se resumen en la Tabla 7.1 y el gráfico asociado.

Tabla 7.1: Soporte de los escenarios en OOWS y OOWS 2.0

Escenario	OOWS 1.0		OOWS 2.0	
	Interacción	Interfaz	Interacción	Interfaz
1. My Home	Si	Si	Si	Si
2. Inbox	Parcial	No	Si	Si
3. My Health	Parcial	Parcial	Si	Si
4. Disease Risk	Si	Parcial	Si	Si
5. Carrier Status	Si	Si	Si	Si
6. Drug Response	Si	Parcial	Si	Si
7. Traits	Si	Parcial	Si	Si
8. Maternal/Paternal Line	Si	No	Si	Parcial
9. Ancestry Painting	Si	No	Si	Parcial
10. Global Similarity	Si	No	Si	Parcial
11. Compare Genes	Si	Parcial	Si	Si
12. Family Inheritance	Parcial	No	Parcial	Parcial
13. 23andMe Community	Si	Parcial	Si	Si
14. Research Surveys	Parcial	Si	Si	Si
15. Research Snippets	Parcial	Parcial	Si	Si
16. Research Initiatives	Parcial	Parcial	Si	Si
17. Research Discoveries	Si	Si	Si	Si
18. Specific Reports	Si	Parcial	Si	Si

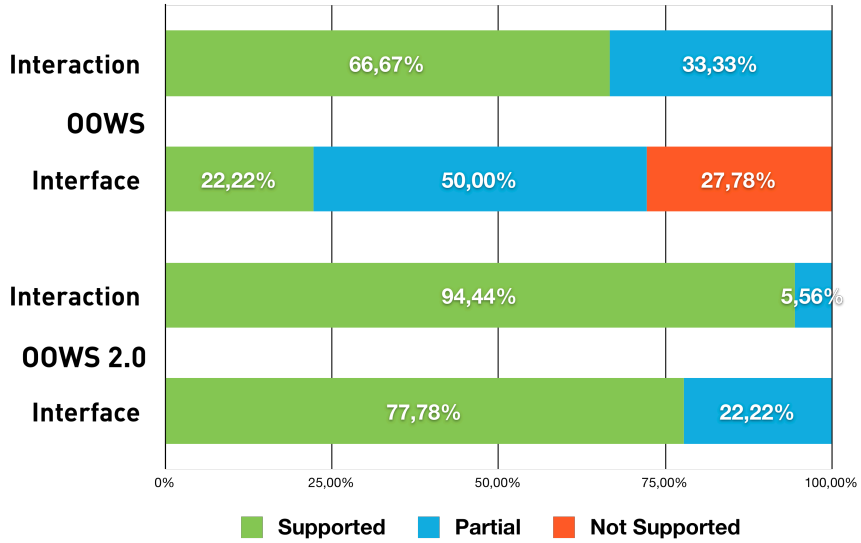


Fig. 7.2: Porcentajes de soporte a los escenarios

En primer lugar, en este análisis no se ha considerado el modelado de la lógica de negocio puesto que ambas aproximaciones comparten el mismo modelo. Respecto al modelado de la interacción, se observa que un amplio número de escenarios son soportados y que no se produce el caso de que ninguno no sea soportado, al menos parcialmente. Este hecho sugiere que los métodos de Ingeniería Web son ampliamente reutilizables, en este aspecto, para el desarrollo de aplicaciones Web 2.0. Por otro lado, el modelo de interacción propuesto en la presente tesis mejora en un 28% la expresividad proporcionada para la interacción. Esta mejora es gracias a, principalmente, la inclusión de AIP como el *Defined List AIP* o el *Summary View AIP*, que permiten la definición completa de varios escenarios.

Respecto al modelado de la interfaz, las contribuciones sí que son claramente significativas. El método OOWS no posee un modelo para representar la interfaz al nivel de detalle necesario por esta razón, en muchos escenarios la interfaz generada no es adecuada. Se aprecia que únicamente el 22'22% de los escenarios son soportados completamente, hecho que implica la necesidad de implementaciones manuales. Las carencias se detectan sobre todo a nivel de eventos de interfaz. *23andMe* es una aplicación Web alta-

mente interactiva, que amolda la visualización de la información acorde a las acciones del usuario sobre la interfaz. Este tipo de comportamiento no es soportado por OOWS. Además, algunos componentes gráficos avanzados tampoco pueden ser especificados utilizando los modelos conceptuales de OOWS.

Esta problemática se soluciona, en gran medida, gracias al Modelo de Interfaz RIA introducido en OOWS 2.0. Como diferencias notables, se mejora el soporte de la interfaz en un 50% y, la interfaz de todos los escenarios es soportada al menos parcialmente. Sin embargo, es difícil conseguir un soporte del 100%, porque algunos de los componentes de interfaz utilizados son específicos de la tecnología RIA utilizada en la aplicación. La expresividad RIA introducida en OOWS 2.0 es toda aquélla que ya soporta la tecnología seleccionada, en nuestro caso *Adobe Flex*, en el proceso de modelado. En el caso que nos ocupa, además de los *widgets* RIA comunes, se han implementado algunos específicos para la visualización de la información genética. Este tipo de *widgets* no se encuentran implementados de manera genérica en la tecnología seleccionada y, por lo tanto, no están soportados a nivel de modelado. Concluyendo se observa una serie de claras mejoras con respecto a los modelos OOWS originales.

7.3 Modelado de la demostración de laboratorio

En esta sección se muestran los modelos necesarios para definir dos de los escenarios previamente analizados: *Compare Genes* y *23andMe Community*. Los escenarios han sido seleccionados porque contienen la expresividad necesaria para ilustrar y evaluar las contribuciones de la presente tesis doctoral. Bajo esta premisa, el primero ha sido seleccionado para ilustrar el uso del modelo RIA, mientras que el segundo muestra, principalmente, la utilidad de los patrones Web 2.0. Para la construcción de los modelos se distinguen tres fases: (1) definición del Modelo de Objetos, (2) definición del Modelo de Interacción Abstracto y (3) definición del Modelo de Interfaz RIA. En las siguientes subsecciones se detallan los modelos que definen estos dos escenarios siguiendo dicho proceso.

7.3.1 Modelado del escenario *Compare Genes*

El escenario *Compare Genes* especifica la interacción para comparar la información en el ámbito de una característica genética o *trait*. La comparación tiene en cuenta los llamados SNP (*Single Nucleotide Polymorphysms)*: variaciones de un único nucleótido en la secuencia de ADN que se producen, por lo menos, en el 2% de la población y que tienen una repercusión documentada sobre un aspecto fenotípico como por ejemplo, el ritmo cardiaco, la resistencia o la fertilidad. La comparación calcula el porcentaje de similitud entre los individuos teniendo en cuenta, únicamente, los SNP asociados a las características genéticas seleccionadas. Además, esta comparación puede realizarse, o bien entre dos individuos, o bien entre un individuo y un conjunto de individuos.

El Modelo de Objetos que abstrae la funcionalidad especificada se muestra en la Fig. 7.3. En este modelo se han omitido las operaciones y atributos que no son relevantes para el análisis. La clase *Subject* define los distintos individuos que participan en la comparación. Cada *Subject* está asociado a varios objetos de la clase *SNPData*, que guardan el valor del individuo para un *SNP* concreto mediante el atributo *variation*. Mediante la clase *Comparison*, se especifican las operaciones para realizar el cálculo que son: *getSingleComparison* y *getManyComparison*. Cada comparación se compone de un sujeto principal, sobre el cual se realiza la comparación (asociación *source*), y de varios sujetos con los cuales se calcula la similitud (asociación *compareTo*). Además, la comparación se realiza teniendo en cuenta una o varias características genéticas representadas por la clase *Trait.* Cada objeto de esta clase se asocia a los SNP que influyen en dicha característica. Adicionalmente, se relaciona cada *Trait* con un conjunto de *Publications,* que proporcionan la validez científica necesaria para avalar dicha característica.

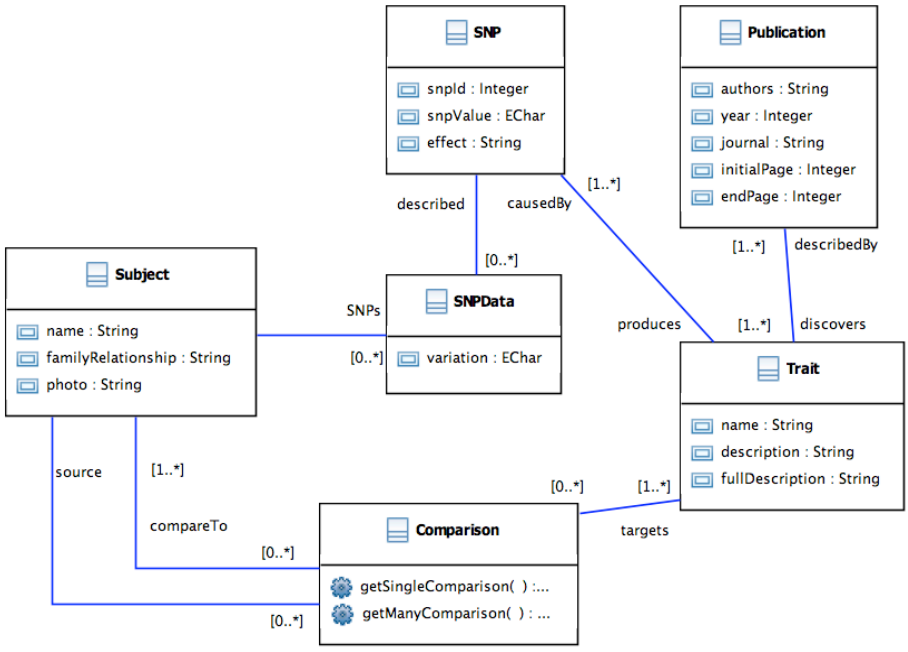


Fig. 7.3: Modelo de Objetos del escenario *Compare Genes*

Una vez realizado el Modelo de Objetos, el siguiente paso es definir el Modelo de Interacción Abstracto correspondiente. En este caso, el escenario *Compare Genes* se especifica como un Contexto de Interacción compuesto de una *Population AIU* y de dos *Service AIU*. En primer lugar, la Fig. 7.4 muestra la *Population AIU,* denominada *Traits Info,* que define una vista de información sobre todos los atributos de las clases *Trait* y *Publication*. Esta AIU incluye un *Summary View AIP* para que, en primera instancia, el usuario tan solo visualice el nombre y la descripción breve de todas las características. La inclusión de este patrón es necesaria para representar, fielmente, la interacción original de *23andMe*.

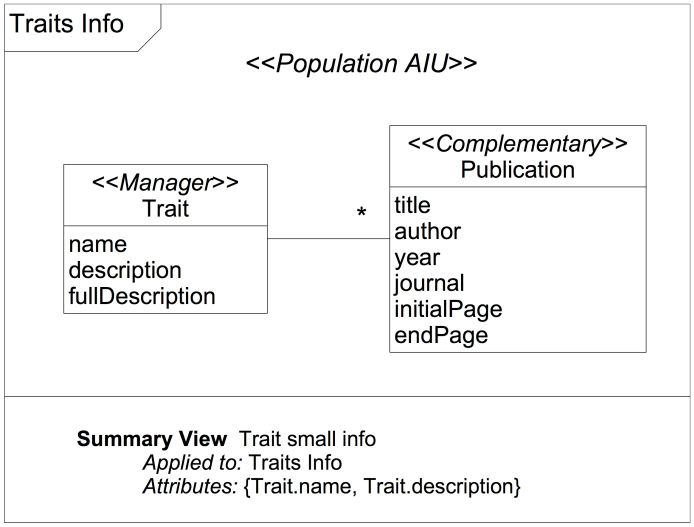


Fig. 7.4: *Population AIU* del contexto *Compare Genes*

Por otro lado, la Fig. 7.5 muestra la especificación de las dos *Service AIU*. La primera de ellas se encarga de modelar la comparación de SNP de un individuo con otro, a partir de un conjunto de características seleccionadas. Esta *Service AIU* se relaciona con la operación *getSingleComparison* perteneciente al Modelo de Objetos. Como argumentos recibe dos objetos de tipo *Subject* y un conjunto de objetos de tipo *Trait,* por lo que la cardinalidad de este argumento se ha definido como *n.* Como resultado, devuelve un conjunto de valores decimales, que indican el porcentaje de similitud, para cada una de las características seleccionadas.

La especificación de la segunda *Service AIU* es similar. En este caso la AIU se relaciona con la operación *getManyComparison*, la cual realiza la comparación de SNP entre varios usuarios simultáneamente. En consecuencia, el argumento *toSubjects* se define con cardinalidad *n* y se establece, como valor por defecto, un conjunto de objetos de tipo *Subject* (en la figura no se muestra la lista completa por motivos de legibilidad). Asimismo, el argumento *Trait* pasa a tener cardinalidad uno puesto que, en esta comparación, solo es posible la comparación de una característica. Por otra parte, el atributo de salida *results* sigue teniendo cardinalidad *n*, pero en este caso, indicando el

porcentaje de similitud de cada usuario de la lista *toSubjects*, para una característica.

Service AIU	Compare Genes One-to-One				
Operation	Comparison.getSingleComparison				
Input Arguments	id	Alias	Type	default	constant
	source Subject	Compare the genome of	Subject: 1	User	false
	toSubject	To the genome of	Subject: 1	none	false
	compare Trait	Select a trait for comparison	Trait: n	Genome Wide Comparison	false
Output Arguments	id	Alias	Type	visibility	
	similarity	comparison result	Decimal: n	true	

Service AIU	Compare Genes One-to-Many				
Operation	Comparison.getManyComparison				
Input Arguments	id	Alias	Type	default	constant
	source Subject		Subject: 1	User	false
	toSubjects		Subject: n	{defined Subjects}	true
	Compare Trait	Select a trait for comparison	Trait: 1	Genome Wide Comparison	false
Output Arguments	id	Alias	Type	visibility	
	subject Results	Percent similarity	Decimal: n	true	

Fig. 7.5: *Service AIU* del contexto *Compare Genes*

Una vez definido el Modelo de Interacción Abstracto, el tercer paso es definir el Modelo de Interfaz RIA. La aplicación de *23andMe* posee una in-

terfaz de usuario muy avanzada creada utilizando un *framework* en *JavaScript*. Este hecho provoca que la interfaz generada por defecto, mediante el método OOWS 2.0, no sea conforme a lo deseado. Para solventar este problema, se debe redefinir el Modelo de Interfaz RIA creado automáticamente. Con dicho fin, se especifican un nuevo conjunto de relaciones de *weaving*, que asocian a las distintas primitivas conceptuales del modelo de interacción de *Compare Genes*, primitivas conceptuales del Modelo de Interfaz RIA más apropiadas. Conviene recordar que en este modelo, no se tienen en cuenta aspectos estéticos de la interfaz, como el color, el tamaño o la tipografía, sino aquéllos componentes de la misma que inciden en la interacción con el usuario.

Tabla 7.2: Modelo de *weaving* para el contexto *Compare Genes*

Primitiva	Id	Widget RIA (Adobe Flex)
Interaction Context	Compare Genes	TabNavigator *Services* + VBox *Traits*
Population AIU	Traits Info	List TraitItems en VBox Traits
Summary View	Trait small Info	Button PlusInfo + Text TraitSummary
Manager Class	Trait	Text TraitDescription
Compl. Class	Publication	Text TraitPublications
Service AIU	Compare Genes One-to-One	Form *Compare* en TabNavigator Services
Service AIU	Compare Genes One-to-Many	Form *CompareMany* en TabNavigator Services
Input Argument	sourceSubject	Autocomplete ComboBox *toSubject*
Input Argument	toSubject	Autocomplete ComboBox *compareSubject*
Output Argument	similarity	List SimiliarityPercentatges
Output Argument	subjectResults	BarChart SubjectsPercentatges

La Tabla 7.2 muestra las relaciones de *weaving* que asocian a cada primitiva conceptual, uno o varios *widgets* pertenecientes a la tecnología RIA *Ado-*

be Flex. Este conjunto de *widgets*, junto con los generados por defecto, conforman el Modelo de Interfaz RIA del escenario. El modelo que define las relaciones de *weaving* se presenta en modo tabular para facilitar su comprensión. El resultado esperado se muestra en la Fig. 7.6, donde se destaca adicionalmente, la correspondencia de la interfaz con las primitivas conceptuales del Modelo de Interacción Abstracto.

Tal y como se aprecia en la Fig. 7.6, el Contexto de Interacción se representa mediante un contenedor tabular (*TabNavigator Services*) compuesto de dos formularios, con el propósito de soportar la entrada de datos de los dos servicios de comparación de SNP. Ambos formularios comparten una caja vertical, en la parte derecha, que contiene una *Population AIU* para visualizar la información de las distintas características genéticas. Esta *AIU*, que se muestra mediante una *widget* de tipo lista (*List TraitItems*), soporta la selección del conjunto de características a comparar. Cada ítem de la lista se compone de tres contenedores de texto (*TraitSummary, TraitDescription* y *TraitPublications*) con la información. El *widget TraitSummary* se asocia a la vista resumida de la AIU mostrando, en primera instancia únicamente, el nombre y la descripción resumida de la característica genética. Cuando el usuario utiliza el botón *PlusInfo*, el resto de la información es desplegada. Esta información detallada se muestra mediante los otros dos *widgets* contenedores de texto que se asocian, respectivamente, a las clases directora y complementarias de la AIU.

El servicio *Compare Genes One-to-One* se compone de dos *widgets* para introducir los sujetos a comparar. El tipo de *widget* utilizado es *Autocomplete ComboBox* que soporta, o bien la introducción del nombre del sujeto, o bien su selección de una lista de nombres que corresponden con el texto introducido. En este caso, el servicio se ejecuta, automáticamente, cuando el usuario selecciona un nuevo sujeto o, cuando se añade una nueva característica genética a la comparación. Los resultados son mostrados en una lista de porcentajes de similitud asociada al *Output Argument similarity.* El funcionamiento del servicio *Compare Genes One-To-Many* es similar al anterior salvo que, la lista de sujetos con los cuales se realiza la comparación es múltiple y se encuentra previamente definida. Los resultados se visualizan mediante un *widget* del tipo *BarChart* que muestra, como un gráfico ordenado, los sujetos con

los cuales se tiene un mayor porcentaje de similitud con dicha característica genética.

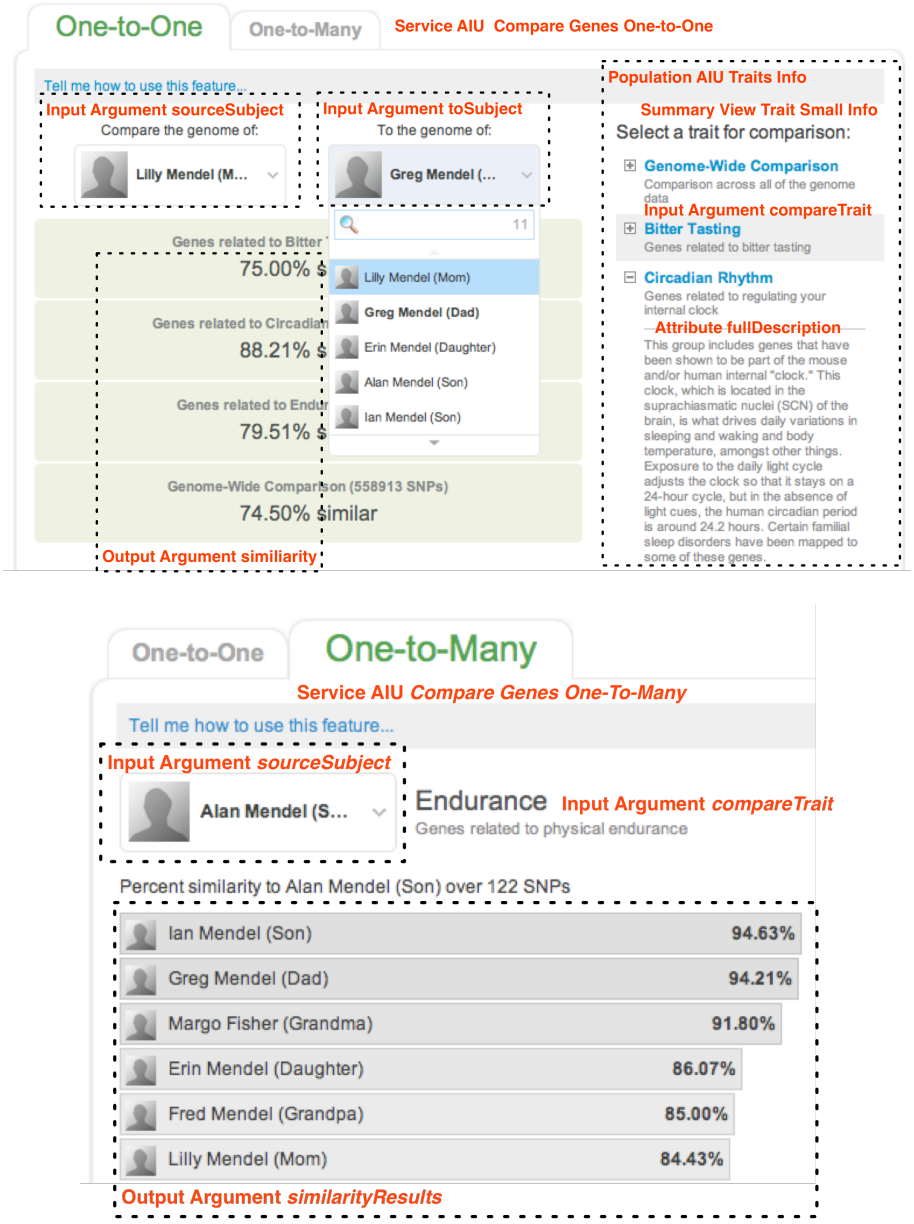


Fig. 7.6: Interfaz RIA del escenario *Compare Genes*

Una vez modelada la parte estática de la interfaz, el siguiente paso de la construcción del Modelo de Interfaz RIA es la definición de las reglas de comportamiento basadas en eventos. Estas reglas se definen utilizando la sintaxis introducida en la sección 5.2.2 de la presente tesis doctoral. En el escenario objeto de estudio, se distinguen los siguientes tres comportamientos de la interfaz originados a partir de eventos:

- Cuando el usuario cambia el sujeto a comparar en alguno de los *widgets* de tipo *Autocomplete,* automáticamente se calcula una nueva comparación entre sujetos. Este comportamiento se encuentra especificado mediante la *Event Rule* de la Fig. 7.7. Básicamente, cuando se produce el evento *itemClick* en el *widget Cbox_ToSubject*, se realiza una reacción de tipo *Invocation* sobre el servicio *Compare Genes One-to-One* encargado de la comparación de SNP. Dicho servicio se encuentra relacionado con el *widget FormCompare,* por lo tanto, es este *widget* el que recibe la reacción. Además, se debe especificar una regla similar para el *widget Cbox_compareSubject,* que contiene el sujeto con el cual comparar. De este modo, independientemente del sujeto que se modifique, se realiza de nuevo la comparación.

```
DEFINE:
    Invocation ChangeSubjectReaction;
SET:
    ChangeSubjectReaction TO FormCompare;
ON:
    Cbox_ToSubject.itemClick(newValue);
DO:
    FormCompare.Invocation();
```

Fig. 7.7: *Event Rule* para recalcular la comparación de características

- Cuando el usuario selecciona una característica genética, ésta se incluye directamente, o bien para calcular la similitud entre los usuarios seleccionados, o bien para calcular la similitud con un conjunto de usuarios. Las dos *Event Rules* mostradas en la Fig. 7.8 especifican este comportamiento. Ambas reglas se definen sobre el evento *itemClick* , perteneciente al *widget TraitItems*, que contiene la lista de las características genéticas que pueden ser comparadas. La

primera regla define, nuevamente, una reacción de invocación sobre el servicio *Compare Genes One-to-One*. Previamente a la invocación, se añade al argumento *compareTrait* del servicio, el identificador de la característica genética seleccionada por el usuario. La segunda regla es necesaria para establecer una condición, que permita la modificación del *widget SimilarityPercentatges,* si la característica genética no está incluida en la comparación. Si la condición se cumple, esta regla modifica dicho *widget*, que muestra los resultados, añadiendo un nuevo ítem para representar el valor de similitud de la nueva característica genética. Esta acción se realiza incrementando la propiedad *items* del *widget* en uno. En este caso se especifica una reacción de tipo *Property Change*.

```
DEFINE:
    Invocation CompareSNPsReaction;
SET:
    CompareSNPsReaction TO FormCompare;
ON:
    TraitItems.itemClick(newValue);
DO:
    compareTrait.values += newValue;
    FormCompare.Invocation();
```

```
DEFINE:
    Change newSimilarityReaction;
IF:
    TraitItems.selectedItem notInList compareTrait.values;
SET:
    newSimilarityReaction TO SimilarityPercentatges;
ON:
    TraitItems.itemClick(newValue);
DO:
    SimilarityPercentatges.items = "SimilarityPercentatges.items+1";
    newSimilarityReaction.Change();
```

Fig. 7.8: *Event Rules* para añadir una nueva característica a la comparación

- La información detallada sobre una característica genética se encuentra por defecto oculta, de tal forma que solo es mostrada, cuando el usuario hace clic en un botón. La *Event Rule* que se define en la Fig. 7.9 especifica dicho comportamiento. La información detallada se encuentra contenida en los *widgets* de texto *TraitDescription* y *TraitPublications*, los cuales se asocian a la

Population AIU correspondiente. En primer lugar, mediante la condición de la regla, se comprueba si la información no está desplegada consultando el estado del botón *PlusInfo*. Seguidamente, la reacción de tipo *Property Change* se asocia a ambos *widgets* de texto y a dicho botón. Cuando se produce el evento *click*, simplemente se establece la propiedad *visible* a cierto en ambos *widgets*, con el objetivo de visualizar la información detallada. Simultáneamente, se cambia el estado del botón para que muestre el signo menos, que indica que la información está desplegada.

```
DEFINE:
    Change displayInfoReaction;
IF:
    PlusInfo.currentState == plus;
SET:
    displayInfoReaction TO TraitDescription;
    displayInfoReaction TO TraitPublications;
    displayInfoReaction TO PlusInfo;
ON:
    PlusInfo.click(button);
DO:
    TraitDespcription.visible = true;
    TraitPublications.visible = true;
    PlusInfo.currentState = minus;
    displayInfoReaction.Change();
```

Fig. 7.9: *Event Rule* para mostrar la información detallada

Estas cuatro reglas se encargan de especificar el comportamiento que se produce en la interacción con el servicio *Compare Genes One-to-One*. Para especificar el comportamiento con el otro servicio, presente en el Contexto de Interacción, basta con descartar la segunda regla de la Fig. 7.8 y, reutilizar el resto de *Event Rules* asociándolas a los eventos correspondientes.

7.3.2 Modelado del escenario *23andMe Community*

El escenario *23andMe Community* implementa la interacción necesaria, para que los usuarios de la aplicación discutan y aporten su opinión sobre temas relacionados con su información genética. El escenario se trata de un foro colaborativo en el cual, las distintas discusiones se clasifican en función de la temática tratada. Como consecuencia, se establece una comunidad de usua-

rios en el marco de la aplicación. Este escenario incluye la clasificación de las discusiones más populares, los *tags* o palabras claves más utilizadas en las discusiones, así como un listado de los usuarios más colaborativos. El modelado de este escenario se va a utilizar con el objetivo de evaluar las ventajas del uso de los patrones Web 2.0 introducidos en la presente tesis. Específicamente, el modelo se centra en la funcionalidad para la creación de nuevas discusiones y la aportación de comentarios a las mismas. La interfaz general de este escenario se observa en la Fig. 7.10.

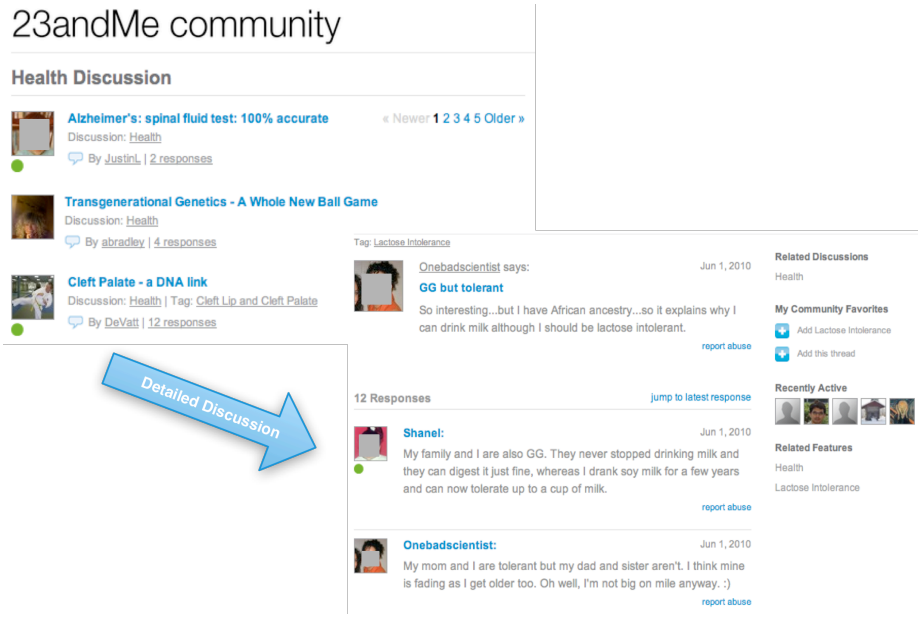


Fig. 7.10: Interfaz del escenario *23andMe Community*

En primer lugar, se ha realizado el Modelo de Objetos que soporta las necesidades de funcionalidad e información de este escenario. Dicho modelo se ilustra en la Fig. 7.11. El modelo se compone de una clase *Discussion*, que clasifica los distintos temas a tratar, y una clase *Thread*, que representa los hilos de discusión. La funcionalidad que ofrece esta última clase es la incorporación de nuevos comentarios al hilo y, la selección de un tema como favorito para que el usuario siga su evolución. Esta ultima funcionalidad implica la creación de una asociación con la clase *User* (*favoriteThreads*), para controlar cuales son los hilos favoritos de cada usuario. Un *Thread* se compone de

un conjunto de respuestas, especificadas mediante la clase *Reply*, también asociadas a un usuario.

Además de la funcionalidad básica para la creación de discusiones, el escenario muestra información de los usuarios así como un enlace a su página personal. La información sobre el usuario se define mediante el conjunto de atributos de la clase *User*. Por otro lado, para la definición de la funcionalidad de la página personal, se utilizan las clases *Public Profile* y *Profile Attribute.* En *23andMe,* la página personal contiene tan solo aquella información que el usuario desea mostrar, pudiendo restringirla a determinados grupos de usuarios. A fin de soportar esta funcionalidad, mediante las instancias de *Profile Attribute* se controla qué atributos de la clase *User* son visibles en el perfil y cuales no. Estos atributos se añaden al perfil a través de la operación *setAttribute*. Por otra parte, para controlar el acceso, se define el atributo *public* a cierto, si no existe ningún tipo de restricción, o se utiliza la operación *grantAccess* para crear relaciones (relación *accessRight*) con los usuarios que tienen acceso al perfil.

A continuación, si analizamos la interacción para especificar el escenario mediante un modelo de OOWS 2.0, se concluye que son necesarias las siguientes AIU:

1. Una *Population AIU* que muestre, de manera indexada, el conjunto de hilos pertenecientes a una temática y las respuestas asociadas al mismo.

2. Una *Service AIU* para la creación de nuevos hilos de discusión.

3. Una *Service AIU* para añadir contestaciones a los hilos de discusión.

4. Una *Population AIU* para consultar el perfil público del usuario.

5. Una *Service AIU* para la creación del perfil público del usuario.

6. Una *Service AIU* para controlar qué atributos pertenecen al perfil y qué usuarios pueden acceder al mismo.

7. Una *Population AIU* que muestre la información resumida de los hilos de discusión favoritos del usuario.

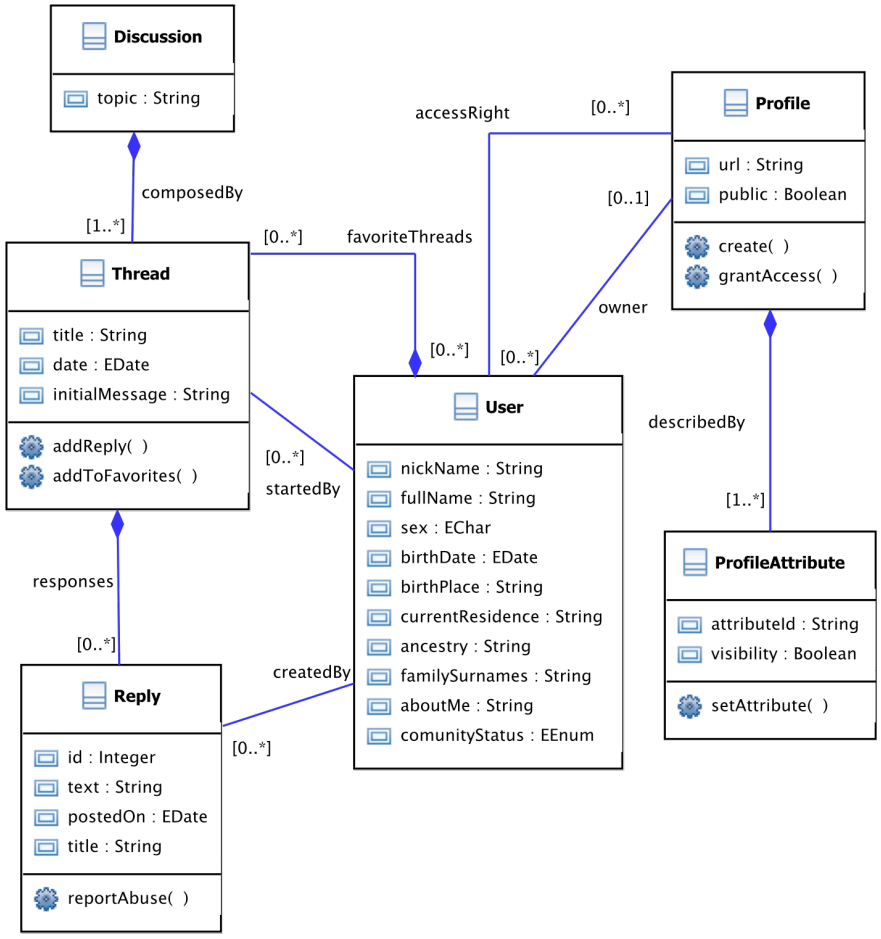


Fig. 7.11: Modelo de Objetos del escenario *23andMe Community*

También deben considerarse los AIP necesarios para establecer la navegación o definir los criterios de ordenación. Se observa que, a pesar que la funcionalidad del escenario no resulta especialmente compleja, es necesario el modelado de un total de siete interacciones. Es más, aunque este tipo de foros de discusión son muy habituales en el marco de las aplicaciones Web sociales, en este caso observamos que una aplicación del dominio bioinformáti-

co también requiere de este tipo de funcionalidad. Con el propósito de simplificar el modelado de este escenario, se han aplicado un conjunto de patrones Web 2.0. El modelo resultante se muestra en la Fig. 7.12.

El Contexto de Interacción *Community Discussion* soporta la funcionalidad, del escenario homólogo de *23andMe,* para la definición de temas de discusión. El contexto se compone, fundamentalmente, de una *Population AIU* y de la aplicación de tres patrones Web 2.0: *Favorite, Quick Comment* y *Public Profile*. La *Manager Class* de la AIU es *Discussion,* que muestra las discusiones sobre un tema en concreto mediante la *Complementary Class Thread.* Sobre esta clase complementaria se aplican dos patrones. En primer lugar, el patrón *Quick Comment* se utiliza para que los usuarios introduzcan nuevas respuestas sobre los distintos *threads*. Si observamos el Modelo de Objetos de la Fig. 7.11, el patrón sustituye el rol que desempeñan la clase *Reply* y la operación *addReply* perteneciente a la clase *Thread.* Analizando la especificación del patrón presentada en la sección 6.2.1, el modelo de funcionalidad de dicho patrón es compatible con la interacción del escenario. Como parámetros del patrón, se define el título de los *Threads* como el identificador sobre el cual se crean los comentarios y, se habilita la opción de responder a los comentarios.

En segundo lugar, también se aplica el patrón *Favorite* sobre la clase *Thread*. Este patrón crea una relación por cada *Thread* que el usuario marca como favorito. Por lo tanto, sustituye a la operación *addToFavorites* y a la asociación *favoriteThreads* del Modelo de Objetos de la Fig. 7.11. Este patrón no implica únicamente la creación de un conjunto de relaciones. El patrón *Favorite* también incluye la especificación de un Contexto de Interacción formado por una *Population* AIU, en el cual el usuario consulta o elimina los distintos hilos que ha añadido como favoritos. De esta forma, la aplicación del patrón evita el modelado de este contexto adicional. Esta funcionalidad se encuentra presente en la aplicación *23andMe,* con la salvedad que es accesible desde la página principal y no, desde el escenario objeto de análisis.

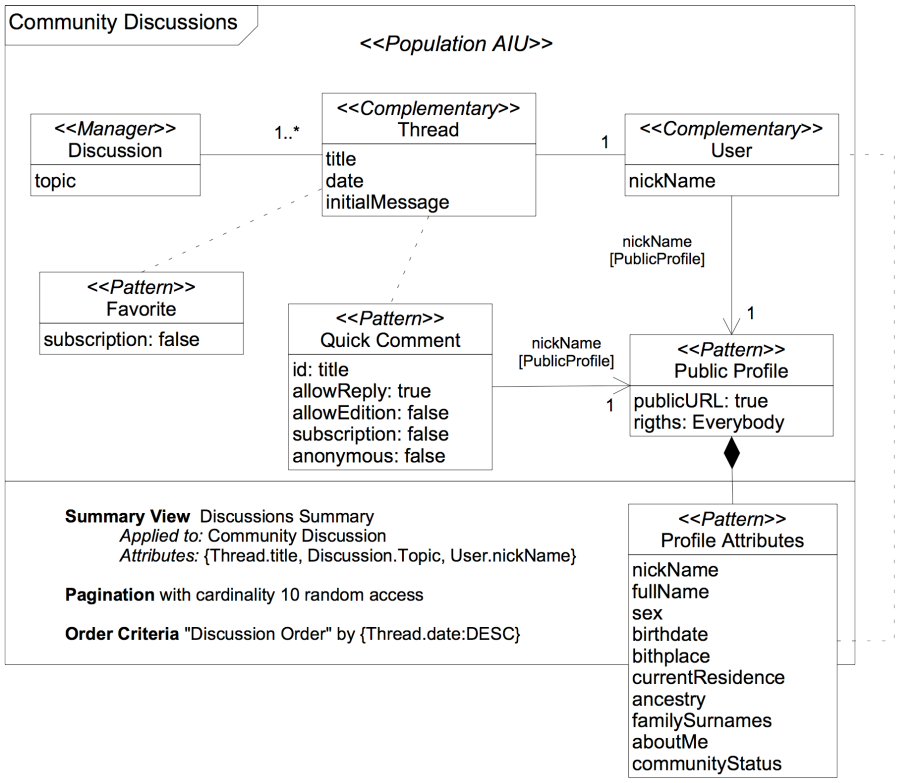


Fig. 7.12: Modelo de Interacción para el escenario *23andMe Community*

El tercer patrón Web 2.0 que se aplica en el contexto es el de *Public Profile*. Este patrón se compone, a diferencia de los dos anteriores, de dos clases estereotipadas que sustituyen, directamente, a las utilizadas para este fin en el Modelo de Objetos de la Fig. 7.11: *Profile* y *Profile Attributes*. La primera de ellas, *Public Profile*, define el perfil propiamente dicho, estableciendo si se dispone de una URL pública, es decir, si el perfil es accesible desde fuera de la aplicación y, el nivel de permiso de acceso que, en este caso, es público para todos los usuarios. La segunda clase, *Profile Attributes*, define una vista sobre los atributos de la clase que representa al usuario en la aplicación, en el ejemplo, la clase *User*. Esta vista selecciona el conjunto de atributos que forman parte de la información del perfil público. En el modelo mostrado, se han incluido todos los atributos, no obstante, la aplicación *23andMe* permite que el usuario escoja qué atributos mostrar.

A diferencia de los otros dos patrones Web 2.0, solo puede existir una instanciación del patrón *Public Profile* en el marco de una aplicación. La razón es que el perfil es único para cada usuario, de ahí que no tenga sentido especificarlo en distintos contextos. A efectos de legibilidad, se ha optado por realizar la especificación del patrón en el contexto modelado, si bien el patrón no tiene por qué, obligatoriamente, asociarse a un Contexto de Interacción determinado. Sin embargo, si es posible hacer referencia al patrón para acceder al perfil en distintos puntos de la aplicación, por ejemplo a través de un *Navigation AIP*. En el modelo de la Fig. 7.12, se utiliza esta clase que representa el patrón, para definir sendos *Object Navigation AIP* desde el patrón *Quick Comment* y desde la clase *User*. Utilizando estas navegaciones, se consulta, respectivamente, el perfil de los usuarios que han creado los comentarios y el perfil del creador del hilo. Por otra parte, la clase *Public Attributes* no se modela como tal, ya que es el usuario y no el analista, quién decide sobre la visibilidad de los distintos atributos de su perfil. Esta selección se realiza a través de una *Service AIU*, presente en el perfil del usuario, que se genera implícitamente al aplicar el patrón. Por esta razón, dicha clase se ha definido fuera del modelo del Contexto de Interacción en la Fig. 7.12, pero se muestra como ejemplo de utilización, la visibilidad por defecto que propone *23andMe*.

En conclusión, la aplicación de los tres patrones reduce la complejidad del modelo a especificar, porque evita la definición de seis AIU. Para soportar completamente la funcionalidad, tan solo sería necesario incluir una *Service AIU* que modelase la operación *Report Abuse*, omitida en el ejemplo. El uso de los patrones simplifica, indirectamente, el Modelo de Objetos propuesto en la Fig. 7.11 puesto que se omiten las clases *Reply, Profile* y *Profile Attribute* y, las dos operaciones de la clase *Thread.* En el análisis propuesto, nos hemos centrado en la funcionalidad relacionada con la creación de discusiones. No obstante, el escenario seleccionado incluye funcionalidad adicional, que también puede ser modelada usando patrones Web 2.0. Por ejemplo, es recomendable la utilización del patrón *Ranking,* para destacar a aquellos usuarios que han participado más activamente en la comunidad. También es posible utilizar el patrón *Tag definition,* para categorizar los distintos temas de discusión planteados.

7.4 Diseño experimental para la evaluación del método

En un trabajo preliminar [Panach, Condori *et al.* 2008] sobre la medición temprana de la usabilidad, en aproximaciones dirigidas por modelos, se evaluó el método OOWS. Los resultados obtenidos en dicho trabajo detectaron graves carencias del método en el terreno de la usabilidad, fundamentalmente, motivados por la carencia de un Modelo de Presentación suficientemente expresivo y, por no poder seleccionar la interacción que era más usable según la percepción de los usuarios.

Siguiendo las mismas directrices que se plantearon en dicho trabajo, esta sección plantea un diseño experimental, que permita evaluar la mejora aportada por las contribuciones de la presente tesis doctoral. Estas contribuciones guardan una estrecha relación con las carencias detectadas en el experimento previo, puesto que surgen para solventarlas. Para realizar una evaluación apropiada del método OOWS 2.0, es necesaria la realización completa de un proceso experimental. Sin embargo, en el estado actual de desarrollo del método, dicha evaluación no ha sido posible. En primer lugar, porque no ha sido implementada una herramienta industrial de soporte al método, puesto que esta tarea queda fuera del alcance de la presente tesis. En segundo lugar, debido a que no es factible encontrar un conjunto de sujetos, con un nivel de conocimiento del método suficiente, para abordar con éxito el desarrollo de un caso de estudio real. A pesar de ello, se considera relevante definir, formalmente, un diseño experimental que permita validar las contribuciones planteadas.

Si bien esta tesis doctoral plantea un método para la producción de aplicaciones Web 2.0, respecto al método OOWS original, principalmente, se introducen mejoras en el nivel de modelado conceptual. Al tratarse de una evaluación de viabilidad, se verifica si los modelos planteados resultan realmente útiles para los analistas o, en otras palabras, si realmente la mejora que aportan justifican su introducción en el marco del método. En función de los resultados de la evaluación, dichos modelos pueden ser mejorados a partir de las carencias detectadas. La utilidad del experimento reside en la realización de un análisis previo, a la costosa implementación final de la herramienta. Cabe reseñar que si un modelo no es capaz de representar de manera adecuada el dominio del problema, en nuestro caso la especificación de aplica-

ciones Web 2.0, la posterior implementación heredará a la fuerza dichas carencias.

Teniendo en cuenta estas ideas, el objetivo del experimento planteado es evaluar la eficiencia temporal y la expresividad conceptual que aportan los modelos realizados utilizando el método OOWS 2.0. La eficiencia temporal se define como el tiempo que analista emplea en la creación de un modelo. Esta característica mide, directamente, la complejidad del proceso de modelado planteado. Por otra parte, entendemos como expresividad conceptual, en el marco de esta tesis doctoral, la fidelidad con la cual un modelo abstrae la realidad que es objeto de su definición. La expresividad conceptual es una característica subjetiva del modelo, determinada bajo el criterio del analista. La evaluación de la expresividad conceptual proporciona el nivel de utilidad de los modelos, para resolver el problema de análisis planteado. Con el propósito de realizar el diseño, se ha seguido el proceso de experimentación descrito por [Wohlin, Runeson *et al.* 2000] en los capítulos 5 y 6, junto con los ejemplos desarrollados por [Travassos, Fabbri *et al.* 2005] que se basan en dicho proceso.

7.4.1 Definición

Objeto de estudio: las nuevas primitivas conceptuales pertenecientes al Modelo de Interacción Abstracto y al Modelo de Interfaz RIA introducidas en el método OOWS 2.0.

Propósito: evaluar la mejora y el impacto temporal en el modelado de aplicaciones Web 2.0 utilizando el método OOWS 2.0 con respecto al método OOWS.

Aspectos de calidad a tratar: eficiencia temporal en el modelado y expresividad conceptual de los modelos realizados.

Perspectiva: investigador que ha desarrollado el método.

Contexto: el estudio se realiza en el marco del centro de Investigación ProS mediante investigadores que tienen conocimientos de desarrollo dirigido por modelos y en particular, del modelado de aplicaciones Web utilizando

OOWS y OO-Method. Todos los sujetos realizan el mismo experimento que consiste en modelar dos escenarios pertenecientes a una aplicación Web 2.0 real. El primer escenario es modelado utilizando el método OOWS, mientras que el segundo es modelado utilizando el método OOWS 2.0.

Preguntas:

P1: ¿La eficiencia a la hora de modelar de los analistas que utilizan el método OOWS 2.0 es mejor que la eficiencia de los analistas que utilizan el método OOWS?

P2: ¿La expresividad conceptual de los modelos realizados mediante el método OOWS 2.0 es mayor que la expresividad conceptual de los modelos realizados mediante el método OOWS?

Métricas:

- **Tiempo de modelado:** número de minutos que el analista tarda en realizar el modelo que representa la interacción con una aplicación Web 2.0. En el caso del método OOWS, este es el tiempo que se tarda en definir el Modelo de Navegación y de Presentación, y en el caso del método OOWS 2.0, el tiempo que se tarda en definir el Modelo de Interacción Abstracto y el Modelo de Interfaz RIA.

- **Requisitos de interacción soportados a nivel de modelado:** nivel de soporte (completo, alto, medio, bajo o nulo) proporcionado por las primitivas conceptuales del método para expresar, mediante un modelo, un conjunto de requisitos de interacción de una aplicación Web 2.0. Esta métrica corresponde a un valor subjetivo otorgado por el analista, a la hora de realizar el modelo que represente dicho requisito.

- **Elementos de interfaz soportados a nivel de modelado:** nivel de soporte (completo, alto, medio, bajo, nulo) proporcionado por las primitivas conceptuales del método para expresar, mediante un modelo, la interfaz de una aplicación Web 2.0. Esta métrica corresponde a un valor subjetivo otorgado por el analista, a la hora de realizar un modelo que represente fielmente la interfaz a desarrollar.

- **Utilización de patrones Web 2.0:** número de patrones Web 2.0 correctamente utilizados, para simplificar el modelo que representa la interacción de una aplicación Web 2.0.

Preguntas no tratadas por el experimento:

- ¿La calidad de las aplicaciones Web 2.0 generadas por el método OOWS 2.0 es mayor que utilizando el método OOWS?

Preguntas abiertas:

- ¿Los analistas entienden adecuadamente y utilizan correctamente el método OOWS 2.0?

- ¿El número de errores de modelado es mayor cuando se utiliza el método OOWS 2.0?

7.4.2 Planificación

Dimensiones del contexto seleccionado

El desarrollo del experimento consiste en el modelado de dos escenarios extraídos de una aplicación Web 2.0. En primer lugar, dichos escenarios no son modelados en un proceso de desarrollo real sino en un entorno controlado. Los sujetos provienen de un entorno académico, debido al estado prematuro de desarrollo en el cual se encuentra el método OOWS 2.0, a nivel de aplicación industrial. Por lo tanto, el experimento se desarrolla en un entorno académico controlado mediante una demostración de laboratorio Por último, los resultados se consideran específicos del método OOWS 2.0, por lo tanto, no se aborda la generalización del experimento en el ámbito de la Ingeniería Web.

Formulación de hipótesis

- *Hipótesis nula* $H1_0$: el tiempo para modelar una aplicación Web 2.0 utilizando el método OOWS es, considerablemente, menor al

tiempo para modelar una aplicación Web 2.0 utilizando el método OOWS 2.0.

- *Hipótesis nula* $H2_0$: en el contexto del modelado de la interacción y la interfaz de una aplicación Web 2.0, la expresividad conceptual de los modelos desarrollados mediante el método OOWS es la misma, que la expresividad conceptual de los modelos desarrollados mediante el método OOWS 2.0.

- *Hipótesis alternativa* $H1_1$: el tiempo para modelar una aplicación Web 2.0 utilizando el método OOWS es igual, al tiempo para modelar una aplicación Web 2.0 utilizando el método OOWS 2.0.

- *Hipótesis alternativa* $H2_1$: en el contexto del modelado de la interacción y la interfaz de una aplicación Web 2.0, la expresividad conceptual de los modelos desarrollados mediante el método OOWS es menor, que la expresividad conceptual de los modelos desarrollados mediante el método OOWS 2.0.

Selección de variables

Variables Independientes:

- Método de modelado utilizado
- Conocimientos de tecnologías RIA
- Conocimiento previo del escenario a modelar

Variables Dependientes:

- Eficiencia del analista en la fase del modelado
- Fidelidad de los modelos respecto al escenario de modelado propuesto

Selección de sujetos

La selección de los sujetos ha sido realizada en el ámbito del centro de investigación ProS por dos razones fundamentales. En primer lugar, los sujetos deben conocer el método OOWS y el método sobre el cual se basa, OO-Method. Si bien el método OO-Method cuenta con una herramienta industrial (O*LIVANOVA*) y es, ampliamente utilizado en entornos industriales, la difusión del método OOWS es todavía muy limitada. Por esta razón, solo pueden seleccionarse sujetos relacionados con el ámbito de investigación de dicho método. En segundo lugar, los sujetos deben tener conocimientos avanzados de modelado conceptual. El centro de investigación ProS posee varias líneas de investigación que hacen uso intensivo de esta disciplina, en diversos dominios de aplicación. No obstante, este tipo de conocimiento todavía no se encuentra extensamente difundido en ambientes industriales, limitando de nuevo los ámbitos disponibles para la selección de sujetos. En conclusión, se ha realizado una selección de sujetos por conveniencia, según la clasificación de [Wohlin, Runeson *et al.* 2000].

Diseño del experimento

A la hora de diseñar el experimento, el factor principal considerado es el método de modelado utilizado. Este factor tiene dos tratamientos: la utilización del método OOWS o la utilización del método OOWS 2.0. Para tener en cuenta dicho tratamiento, en primer lugar los analistas modelan utilizando OOWS, un escenario que representa un fragmento de la interacción del usuario con una aplicación Web 2.0. Posteriormente, repiten dicha tarea de modelado mediante el método OOWS 2.0.

Un aspecto a considerar en el experimento es si los sujetos poseen el nivel de conocimiento suficiente, tanto del método OOWS como de modelado conceptual, para abordar las tareas propuestas. Para evitar que ambos parámetros, conocimiento del método y experiencia de modelado, influyan en el experimento, se realiza una selección previa de sujetos. Como consecuencia, el nivel de conocimiento de los sujetos se considera constante. Debido al estado de desarrollo del método OOWS 2.0, ninguno de los sujetos tiene el nivel de conocimiento requerido de este método. Por lo tanto, es necesario un entrenamiento previo antes de abordar las tareas de modelado utilizando

OOWS 2.0. Este entrenamiento se imparte una vez se hayan realizado los modelos mediante OOWS, con el fin de evitar que el conocimiento de los nuevos modelos influya en el resultado.

En el desarrollo del experimento existen dos factores, conocimiento previo del escenario a modelar y experiencia con tecnologías RIA, que pueden influir en el resultado pero cuyo efecto no es de relevancia en el estudio. Para omitir ambos factores, se ha optado por aplicar el principio de bloqueo (*blocking*), definido por [Wohlin, Runeson *et al.* 2000], tal y como se detalla a continuación:

- ***Conocimiento previo del escenario a modelar***: si en ambos tratamientos utilizamos el mismo escenario de modelado, es de esperar que el tiempo de modelado se vea reducido en el segundo tratamiento. La causa es que en un proceso de modelado conceptual, el razonamiento sobre las distintas entidades que componen el modelo es una actividad con un alto impacto temporal. Ya que el método OOWS y OOWS 2.0 se basan en el mismo Modelo de Objetos, el hecho de haber realizado un modelo previo impactaría, positivamente, en el tiempo de modelado utilizando OOWS 2.0. Para solventar este problema, se ha optado por definir en el experimento dos escenarios de modelado diferentes. De esta forma, el conocimiento previo no afecta al resultado porque el analista debe volver a razonar todo el modelo conceptual desde cero.

- ***Experiencia con tecnologías RIA***: El método OOWS 2.0 introduce un Modelo de Interfaz RIA el cual, se encuentra muy ligado con este paradigma tecnológico. Dado que en el entrenamiento solo se aborda el uso del modelo, los sujetos que tengan conocimientos previos de estas tecnologías son susceptibles a entender mejor dicho modelo y, en consecuencia, a generar modelos conceptuales de mayor expresividad. Por lo tanto, este factor afecta tanto al tiempo de modelado como a la expresividad conceptual. Para minimizar el efecto de este factor, en primer lugar, se captura la experiencia de los sujetos en este tipo de paradigma tecnológico, clasificándolos en dos grupos en función de sus conocimientos: grupo no-RIA y grupo RIA. Además, como se proponen dos escenarios de modelado, se ha

decidido que en uno de ellos el impacto de utilizar el Modelo de Interfaz RIA sea nulo, para que sea modelado por el grupo no-RIA.

Para tener en cuenta el bloqueo simultaneo de ambos factores, se diseña un escenario en el cual el Modelo de Interfaz RIA es relevante (Escenario RIA) y otro escenario (Escenario P), en el cual el uso de los patrones Web 2.0 es relevante, pero el modelo de la Interfaz RIA no es necesario. En ambos escenarios, se considera relevante el Modelo de Interacción Abstracto. Debido a que OOWS no contempla un equivalente al Modelo de Interfaz RIA, el escenario RIA es modelado, primeramente, por el grupo No-RIA. Por lo tanto, ya que los sujetos no deben repetir el escenario a modelar con el otro método, modelarán mediante OOWS 2.0 un escenario en el cual el Modelo de Interfaz RIA no tiene que ser modificado. Para el grupo RIA, se invierte el orden de modelado de los escenarios con el fin de desarrollar, necesariamente, un Modelo de Interfaz RIA mediante OOWS 2.0. Teniendo en cuenta este bloqueo y el factor a considerar, la Tabla 7.3 resume el diseño final del experimento a realizar:

Tabla 7.3: Diseño del experimento

Método utilizado	Grupo No RIA	Grupo RIA
OOWS	Escenario RIA	Escenario P
Entrenamiento en OOWS 2.0		
OOWS 2.0	Escenario P	Escenario RIA

Amenazas a la validez de los resultados

En este apartado, se analiza la validez de los resultados en función de los cuatro tipos de amenazas propuestas por [Cook & Campbell 1979]. Para cada amenaza, se detalla la posible influencia que tiene en el experimento y cómo ésta ha sido minimizada:

- **Amenazas de conclusión**: la validez del experimento propuesto es altamente dependiente de la fiabilidad de las métricas utilizadas. En nuestro caso, se han utilizado dos métricas, de carácter subjetivo, como son el grado de soporte a los requisitos de interacción e interfaz a nivel de modelado. En consecuencia, es necesario

considerar esta subjetividad, puesto que un modelo que soporta adecuadamente los requisitos según la percepción de un analista, no tiene por qué ser suficiente expresivo para otro. Por esta razón, también se tiene en cuenta el nivel de experiencia de los analistas en los métodos, así como el modelo resultante que proporcionan en el experimento.

- **Amenazas internas**: una amenaza interna del experimento es la repetición del mismo escenario de modelado, debido a que las experiencias en el modelado pueden ser intercambiadas entre los dos grupos. Esta amenaza se evita separando los grupos en dos aulas distintas, de tal forma que no tienen conocimiento de las tareas de modelado que ha realizado el otro grupo. El hecho que los escenarios no sean modelados a la misma vez por todos los sujetos, no se considera un factor suficientemente relevante, para afectar a la validez de los resultados. Por otro lado, ya que ambos grupos tienen que realizar el modelado utilizando ambos métodos, se debe evitar que la explicación de alguno de ellos resulte más atrayente a los usuarios, afectando a su opinión sobre el mismo.

- **Amenazas de construcción**: existe una amenaza de construcción estrechamente relacionada con la amenaza de conclusión presentada. Puede ocurrir que los sujetos no comprendan adecuadamente la métrica relacionada con la expresividad conceptual, es decir, cuando un modelo representa fielmente el escenario descrito. Esta circunstancia supone un grave problema para la validez. Para solventarla, a lo largo del entrenamiento con el método OOWS 2.0, se utilizan ejemplos de modelado tanto correctos como incorrectos para aclarar este criterio a los sujetos.

- **Amenazas externas**: este tipo de amenazas limitan la capacidad de generalizar los resultados del experimento a un entorno industrial. El objetivo de este experimento no es alcanzar dicha generalización, por lo tanto, su análisis no es relevante.

7.4.3 Instrumentación

Para la realización del experimento, se define un conjunto de instrumentos detallados en esta sección:

Cuestionario Previo

Este primer cuestionario utilizado en el experimento se utiliza para capturar la experiencia de los sujetos en los métodos de modelado OOWS y OO-Method y, en el uso de tecnologías RIA. El cuestionario consiste en dos preguntas con un conjunto de respuestas prefijadas, de entre las cuales, los sujetos tienen que seleccionar la opción que más se ajusta a su nivel de conocimiento. Las preguntas de dicho cuestionario son:

CP1: ¿Cuál es su experiencia en el modelado de aplicaciones Web mediante OOWS y OO-Method?

1. No he utilizado ningún método para el modelado de aplicaciones Web.

2. Solo he utilizado OO-Method para el modelado de aplicaciones Web.

3. He utilizado OOWS para el modelado de aplicaciones Web.

CP2: ¿Cuál es su nivel de conocimiento sobre las tecnologías RIA?

1. No conozco ninguna.

2. Conozco alguna tecnología, pero no la he utilizado para el desarrollo aplicaciones Web.

3. He utilizado algún tipo de tecnología RIA para el desarrollo de aplicaciones Web.

4. Utilizo habitualmente tecnologías RIA para el desarrollo de aplicaciones Web.

La primera pregunta es utilizada para descartar la participación de determinados sujetos en el experimento, debido a que sus conocimientos no son los adecuados. En concreto, si el sujeto selecciona la opción 1 de la pregunta CP1 queda automáticamente descartado. Aquellos usuarios que tengan experiencia previa con OO-Method se les considera válidos, puesto que muchas de las primitivas conceptuales de OOWS 2.0 provienen de dicho método. Por otra parte, la segunda pregunta es utilizada para realizar una clasificación previa de los sujetos. Aquéllos que seleccionen la opción 1 o 2 pertenecerán al grupo de no-expertos en tecnologías RIA.

Especificación de los escenarios de modelado

El segundo instrumento del experimento es una especificación detallada de los modelos que los sujetos tienen que realizar. Esta especificación se basa en dos escenarios seleccionados de una aplicación Web 2.0 real. En concreto para la realización del experimento se proponen los mismos escenarios utilizados en las secciones 7.3.1 y 7.3.2: *Compare Genes* y *23andMe community*.

La especificación consta, en un primer lugar, de una descripción textual del objetivo y funcionalidad que representa el escenario, incidiendo en la terminología del dominio que sea necesaria para que el sujeto comprenda, adecuadamente, que debe modelar. Esta descripción del escenario es similar a la utilizada anteriormente en este capítulo. Seguidamente, se presenta una visión general de la interfaz y se les proporciona a los sujetos una URL, para que interactúen ellos mismos con la aplicación Web 2.0 real. El tercer elemento de esta especificación es un conjunto especifico de tareas a modelar pertenecientes al escenario. Mediante la acotación de la funcionalidad del escenario a modelar, se pretende que los sujetos se enfrenten a los mismos problemas de modelado, eliminar posibles ambigüedades en la interpretación y obtener como resultado modelos más homogéneos. Las tareas de modelado de cada escenario son:

Compare Genes

T1: modelar la lista de selección de las características genéticas a comparar. Cuando el usuario seleccione una característica, ésta se debe añadir automáticamente al servicio de comparación y recalcularse el resultado. Además, tiene que modelarse la información detallada de la característica,

descripción extendida y publicaciones que la justifican, que aparece cuando se hace clic en el botón desplegable.

T2: modelar el servicio de comparación de SNP *One-to-One*. La interacción debe considerar que cada vez que se cambie uno de los usuarios, se recalcule de nuevo el resultado de la comparación. Además, tiene que contemplarse, en la introducción de los usuarios a comparar, el uso del mecanismo de autocompletado.

23andMe Community

T3: modelar la visualización de los últimos hilos de discusión creados por los usuarios, junto con la navegación para acceder a los distintos comentarios.

T4: modelar la creación de nuevos hilos de discusión. La creación de hilos tiene que contemplar la inclusión de los mismos en una temática definida previamente. La creación de un nuevo hilo de discusión implica la creación de un mensaje inicial, cuyo autor es el creador del hilo.

T5: modelar la visualización de las distintas respuestas creadas en un hilo. Se tiene que incluir el modelado de un servicio, para que los usuarios respondan desde dicha visualización.

T6: modelar el servicio para añadir un hilo a los favoritos del usuario. El modelo también tiene que expresar la visualización de los hilos que el usuario ha seleccionado como favoritos.

Tutorial del método OOWS 2.0

Los sujetos seleccionados no tienen conocimientos avanzados en el método OOWS 2.0 puesto que es introducido, por primera vez, en la presente tesis doctoral. Por lo tanto, antes de realizar el modelado mediante el método OOWS 2.0, se tiene que realizar un entrenamiento previo. Para la realización de este entrenamiento, se utiliza un tutorial compuesto por una serie de transparencias en las cuales se introduce, de manera guiada, los modelos de OOWS 2.0 que son utilizados en el experimento. Además de introducir las primitivas conceptuales de los modelos, el tutorial muestra ejemplos reales de modelado utilizando el método. Este tutorial es impartido en un

seminario de dos horas de duración, más una sesión opcional de media hora para la aclaración tutelada de dudas. El material impartido sobre el método OOWS 2.0 se pone a la disposición de los sujetos a lo largo del experimento, con el fin de poder consultarlo si es necesario.

Cuestionario de evaluación del proceso de modelado

Este instrumento es utilizado para realizar la evaluación del experimento. El cuestionario se compone, en primer lugar, de una serie de preguntas identificativas. Antes de realizar las tareas de modelado, el sujeto tiene que escribir el grupo al cual está asignado y el escenario correspondiente de modelado que va a especificar (*Compare genes* o *23andMe community*). A continuación, el sujeto debe anotar la hora de inicio de la tarea de modelado y, posteriormente, la hora de finalización. Mediante esta pregunta (A0), se controla la métrica del tiempo de modelado.

Como siguiente paso, se realizan una serie de afirmaciones en positivo, sobre distintos aspectos de la tarea de modelado, con tal de capturar la opinión del sujeto sobre la utilización de los métodos. El sujeto proporciona su opinión siguiendo una escala *likert* de 1 a 5 en función de si: (1) "Está totalmente de acuerdo con la afirmación"; (2) "Está bastante de acuerdo con la afirmación"; (3) "No esta de acuerdo, pero tampoco en desacuerdo con la afirmación"; (4) "Está bastante en desacuerdo con la afirmación"; (5) "Está completamente en desacuerdo con la afirmación".

El cuestionario se compone de un conjunto de afirmaciones, acorde con dicha escala, que tienen que ser rellenadas para ambos escenarios/métodos. Por un lado, las afirmaciones A1-A6 son dependientes del escenario seleccionado, porque hacen referencia a una tarea de la especificación anteriormente introducida. Por otra parte, las afirmaciones A11-A14 únicamente son contestadas cuando se aplica el método OOWS 2.0. A continuación, se detallan estas afirmaciones:

Generales

A1-A6: "Las primitivas conceptuales del método me han permitido especificar un modelo que expresa fielmente la tarea de modelado *Tn*".

En esta afirmación *n* hace referencia a las tareas de la T1 a la T6 definidas en la especificación de los escenarios.

A7: "Las primitivas conceptuales del método me han permitido especificar un modelo que representa fielmente los componentes de interfaz del escenario".

A8: "Las primitivas conceptuales del método me han permitido especificar los cambios que se producen en la interfaz del escenario, como consecuencia de la interacción del usuario (por ejemplo, el clic de un botón)".

A9: "Considero que el método OOWS/OOWS 2.0 es una forma eficiente de abordar el modelado del escenario propuesto".

A10: "Considero que el método OOWS/OOWS 2.0 es lo suficientemente expresivo para modelar la interacción escenario propuesto".

En las preguntas A9 y A10 el usuario tiene que considerar el método que está utilizando

Específicas para OOWS 2.0

A11: "La definición del Modelo de Interfaz RIA me ha resultado sencilla".

A12: "El Modelo de Interfaz RIA ha sido útil para modelar el escenario".

A13: "Los patrones Web 2.0 propuestos por el método han sido útiles para resolver el escenario".

L1: Enumera los patrones Web 2.0 utilizados si procede.

En esta última pregunta el sujeto escribe el nombre de los patrones que haya utilizado en el escenario.

Por último, cada una de las respuestas del cuestionario guarda una relación directa con una de las métricas utilizadas en el experimento. La siguiente tabla muestra, a modo de resumen, esta relación:

Tabla 7.4: Relación entre las métricas y el cuestionario de evaluación

Métrica	Respuestas
Tiempo de modelado	P1, A9
Requisitos de interacción soportados a nivel de modelado (Escenario 1)	A1-A2, A10
Requisitos de interacción soportados a nivel de modelado (Escenario2)	A3-A6, A10
Elementos de interfaz soportados a nivel modelado	A7-A8, A11-A12
Uso de patrones Web 2.0 (Escenario 2)	A13, L1

7.4.4 Proceso

A continuación, se detalla el desarrollo del experimento describiendo las distintas sesiones que lo componen. El experimento se realiza en tres sesiones, cada una de ellas en un día distinto, tal y como resume la Tabla 7.5.

Tabla 7.5: Orden temporal de las tareas

	Tareas	Duración
Sesión 1	1 a la 5	1:30 horas
Sesión 2	6 a la 9	2:00 horas
Sesión 3	10	45 minutos

Sesión 1

1. **Presentación del experimento** (30 minutos): en primer lugar, se presenta a los sujetos los objetivos generales del experimento a realizar. Seguidamente, se realiza una pequeña presentación de la aplicación Web 2.0 que utilizarán: *23andMe*. En esta presentación se resolverán aquellas dudas que surjan sobre el dominio bioinformático de la aplicación.

2. **Realización del cuestionario previo** (5 minutos): después de la presentación, se utiliza el cuestionario previo con el fin de clasificar los sujetos en dos grupos, en función de su dominio de las tecnologías RIA. Los grupos de sujetos permanecerán separados desde este punto.

3. **Presentación del escenario de modelado con OOWS** (20 minutos): Se presenta el escenario de modelado correspondiente, a cada uno de los grupos, mostrando la interacción e interfaz que tienen que modelar con OOWS. A continuación, se entrega la especificación de las tareas de modelado correspondientes y, se solucionan en común las posibles dudas que surjan en la lectura de la especificación. El grupo RIA modelará el escenario correspondiente a la sección *23andMe Community*, mientras que el grupo no-RIA modelará la sección *Compare Genes*.

4. **Modelado utilizando OOWS** (máximo 30 minutos): Cada uno de los grupos realiza el modelo de OOWS correspondiente que se compondrá tanto del Modelo de Navegación como del Modelo de Presentación. Este modelo se realiza utilizando el editor correspondiente y entregándolo a su finalización.

5. **Cumplimentar el cuestionario de evaluación de OOWS** (5 minutos): conforme los sujetos terminen el modelo, procederán a cumplimentar el cuestionario de evaluación correspondiente.

Sesión 2

6. **Entrenamiento en OOWS 2.0** (una hora): la segunda sesión comienza con un seminario explicativo del método OOWS 2.0. A través de una presentación con transparencias, se incidirá en las diferencias con el método anterior y se mostrarán modelos de ejemplo, que resalten las nuevas contribuciones del método.

7. **Presentación del escenario de modelado con OOWS 2.0** (20 minutos): Esta tarea es similar a la de la primera sesión, con la salvedad de que a cada grupo se le presenta el escenario que todavía no ha modelado.

8. **Modelado utilizando OOWS 2.0** (máximo 40 minutos): Al igual que en la sesión anterior, los sujetos procederán a crear los modelos correspondientes. En esta tarea, tiene que definirse tanto un Modelo de Interacción Abstracto como, opcionalmente, un Modelo de Interfaz RIA compuesto de las relaciones de *weaving* correspondientes. Se les proporciona más tiempo a los sujetos dada la mayor complejidad del modelo a realizar.

9. **Cumplimentar el cuestionario de evaluación de OOWS 2.0** (5 minutos): se repite de nuevo el cuestionario de evaluación. Como diferencia, se recalca a los sujetos que tienen que responder a las preguntas que no estaban en el cuestionario anterior.

Sesión 3

10. **Discusión final** (45 minutos): en esta última sesión, se discute con los usuarios sus experiencias e impresiones sobre el experimento y, la utilización de los diferentes métodos. Se muestran los resultados finales y se pregunta su opinión sobre los mismos. De esta forma, se busca obtener retroalimentación de cara a resolver posibles defectos en el experimento realizado o, en el propio método OOWS 2.0.

7.5 Conclusiones

En este capítulo se han realizado tres tareas con el objetivo de plantear la viabilidad del método OOWS 2.0, en el marco del modelado de aplicaciones Web 2.0: nivel de soporte de requisitos de modelado conceptual, modelado de un conjunto de escenarios y diseño de una evaluación experimental a priori. Como demostración de laboratorio para realizar la evaluación, se ha utilizado en dichas tres tareas la misma aplicación Web 2.0. Las lecciones aprendidas a lo largo de esta experiencias han sido las siguientes:

- El análisis del soporte a nivel de modelado, ha puesto de manifiesto una clara mejora en la aplicación analizada mediante la utilización del método OOWS 2.0. Esta mejora resulta notable a la hora de

modelar la interfaz de usuario, consiguiendo un 50% adicional de escenarios modelados.

- La utilización de los patrones Web 2.0 simplifican el modelo final resultante. De esta manera se consigue una notación más comprensible y fácil de entender por parte del analista además de mejorar su eficiencia.

- Debido a limitaciones temporales y de recursos no ha sido posible llevar a cabo la ejecución de la evaluación experimental para demostrar de forma más precisa las mejoras que introduce el método. Sin embargo, se considera relevante describir con detalle el experimento a realizar para conseguir estos resultados.

Las hipótesis iniciales, por demostrar, sugieren que la utilización del método OOWS 2.0 implica una mayor complejidad en los modelos conceptuales que incide, negativamente, en la eficiencia de los analistas. Esta complejidad es consecuencia de la introducción del Modelo de Interfaz RIA, el cual incluye la definición de reglas de *weaving* y de eventos de interfaz. A pesar de ello, este decremento de la eficiencia se ve recompensado por un claro aumento de la expresividad conceptual, cuando el objetivo es desarrollar aplicaciones Web 2.0. La realización de este experimento será abordada, en el marco de un proyecto bioinformático, con el objetivo de desarrollar un portal Web 2.0 para la generación de informes genéticos. Este portal se encuentra actualmente en construcción, aplicando en su desarrollo, parte de las contribuciones del método OOWS 2.0.

Capítulo 8

Conclusiones

En este capítulo se desarrollan las conclusiones finales de la presente tesis. En primer lugar, la sección 8.1 resume las contribuciones principales que se han desarrollado a lo largo de la tesis doctoral. A continuación, la sección 8.2 presenta el trabajo que se está desarrollando actualmente en el marco del método OOWS 2.0 y las líneas de trabajo futuras. La sección 8.3, detalla las distintas publicaciones académicas realizadas en el marco de la tesis doctoral. Por último, en la sección 8.4 se plantea la reflexión final de la tesis doctoral.

8.1 Contribuciones principales

A lo largo de la presente tesis, se ha justificado la necesidad de incluir la Web 2.0 en el ámbito de la Ingeniería Web. La principal razón es que el desarrollo de aplicaciones Web 2.0 es una disciplina compleja, que requiere de herramientas, modelos y métodos eficaces. La contribución fundamental de la presente tesis es la definición, mediante modelos conceptuales, del soporte necesario para el modelado avanzado de la interacción en el marco de las

aplicaciones Web 2.0. El resultado más destacado de la aplicación de estos modelos es el nuevo método de Ingeniería Web OOWS 2.0. En el ámbito de la Ingeniería Web, esta es la primera propuesta que combina la expresividad necesaria para el modelado, tanto de la interacción como de la Interfaz RIA, de una aplicación Web 2.0. Además, al tratarse de una evolución del método OOWS, mantiene la meta de la generación sistemática del código de la aplicación Web 2.0 modelada. Han sido varias las contribuciones presentadas en esta tesis para alcanzar dicho objetivo general. Específicamente, las contribuciones a nivel de modelado conceptual han sido las siguientes:

- La **definición de un Modelo de Interacción Abstracto**, basado en los modelos de Navegación y Presentación de OOWS y el Modelo de Presentación de OO-Method. A partir del análisis de las similitudes y diferencias entre ambos, se ha definido un modelo que continua siendo abstracto, es decir, independiente de la tecnología a utilizar y, que combina las mejores prácticas metodológicas de ambos métodos. Este nuevo modelo se fundamenta en las primitivas conceptuales previamente definidas, para proporcionar una propuesta original y evolutiva, que permite el modelado preciso de la interacción de una aplicación Web 2.0. La principal ventaja de esta aproximación es que, gran parte de la semántica de este modelo ha sido validada, indirectamente, a través de los diversos casos de estudio realizados en el ámbito de OO-Method y OOWS. Además, enfatiza el modelado de la interacción, como un elemento de primer orden en el desarrollo de aplicaciones Web 2.0.

- La **definición de un metamodelo para la especificación de Modelos de Interfaz RIA**. Esta contribución resuelve la necesidad de soportar las distintas tecnologías que implementan la interacción abstracta, como una interfaz de usuario. En concreto, el metamodelo trata el modelado de interfaces de usuario, que son implementadas mediante una tecnología basada en el paradigma RIA. Para conseguir este objetivo, se ha propuesto un metamodelo genérico que recoge la expresividad, a nivel de interfaz, que es común en el marco de estas tecnologías. Este metamodelo es posteriormente especializado en una tecnología RIA concreta, para explotar las características tecnológicas propias que ofrece. Esta es una ventaja determinante de

la contribución puesto que aporta una gran riqueza conceptual, a la hora de especificar la interfaz de una interacción. Como ejemplo, en la presente tesis, se ha definido un metamodelo de interfaz para la tecnología *Adobe Flex*. Además, la propuesta también incluye la expresividad necesaria, a nivel de modelado conceptual, para definir el comportamiento dirigido por eventos, tan habitual en el ámbito de las RIA, a través de reglas. En este sentido, se propone una sintaxis textual, basada en el lenguaje XText, para simplificar la definición de estas reglas de comportamiento.

- Se ha definido el **concepto de patrón Web 2.0 a nivel de modelado**, el cual representa una solución a un problema de interacción habitual en el desarrollo de aplicaciones Web 2.0. Para la obtención de un conjunto de patrones Web 2.0 representativo, se ha realizado un análisis de quince aplicaciones Web 2.0. Este análisis ha determinado qué patrones se aplican con mayor asiduidad. Con el objetivo de representar estos patrones Web 2.0 como modelos conceptuales, se han seleccionado notaciones ampliamente utilizadas como el diagrama de clases UML, para modelar su funcionalidad, y la notación CTT, para describir la interacción. La ventaja de está aproximación es que los patrones se definen independientemente de un método de Ingeniería Web específico.

Además de la definición de los modelos conceptuales, se ha abordado su integración, especificación y evaluación en el ámbito de la Ingeniería Web. Las contribuciones realizadas en este sentido han sido las siguientes:

- Se ha definido **una estrategia basada en metamodelos de *weaving* para relacionar los modelos de interacción con los modelos de interfaz.** Esta estrategia genérica ha sido aplicada en el marco del método OOWS 2.0. Hasta el momento, cada primitiva conceptual de OOWS estaba ligada a un componente de interfaz específico, a nivel de implementación. Mediante la definición de un modelo compuesto de relaciones de *weaving*, el analista es capaz de definir para cada primitiva conceptual del Modelo de Interacción Abstracto, qué primitiva del Modelo de Interfaz RIA es la más adecuada. En consecuencia, se le otorga al analista la capacidad de definir

interfaces usando los *widgets* que considere más usables para la interacción a representar. La ventaja de esta aproximación es la introducción de una mayor flexibilidad en el proceso de modelado. Como resultado adicional, se ha definido un conjunto de relaciones de *weaving* para la generación, a partir de un Modelo de Interacción Abstracto, de un Modelo de Interfaz RIA en la tecnología *Adobe Flex*.

- Se ha definido **una estrategia de transformación modelo a modelo para la inclusión de los patrones Web 2.0 en los métodos de Ingeniería Web**. Esta estrategia detalla las modificaciones necesarias, a nivel de metamodelo, para incluir la definición de un patrón Web 2.0. Además, formaliza la representación del patrón, en el marco de un método de Ingeniería Web específico, mediante un lenguaje de transformación de modelos, en nuestro caso ATL. Como ejemplo ilustrativo, se han integrado, siguiendo esta estrategia, los patrones *Quick Comment* y *Notification* en el método OOWS 2.0.

- Se han generado una **versión preliminar de los editores de los modelos conceptuales propuestos**. Las diversas contribuciones presentadas a nivel conceptual, han sido formalizadas mediante la construcción de los metamodelos correspondientes. Estos metamodelos han sido especificados utilizando Eclipse EMF, en concreto, mediante su descripción utilizando el lenguaje de metamodelado *Ecore*. De este modo, ha sido posible generar, automáticamente, a partir de dichos metamodelos un editor visual para la creación de los modelos. Por otro lado, los metamodelos de carácter textual han sido descritos usando una gramática basada en el lenguaje XText. El uso de esta tecnología, también incluida en el entorno Eclipse, ha permitido la generación de un editor textual para la creación de los modelos.

- Como paso previo a la construcción del soporte tecnológico del método, se ha realizado una **evaluación de la viabilidad de las distintas contribuciones**. Para llevar a cabo este objetivo, se ha utilizado como demostración de laboratorio un conjunto de escenarios extraídos de la aplicación Web *23andMe*. En primer lugar,

se ha analizado en que medida la aplicación del método OOWS 2.0, da un soporte mayor en el modelado de la interacción y de la interfaz de las aplicaciones Web 2.0, con respecto a OOWS. En segundo lugar, se han modelado dos escenarios representativos de la aplicación utilizando los nuevos modelos propuestos por el método OOWS 2.0. Por último, se ha planteado el diseño de un experimento para evaluar la mejora, en términos de eficiencia y expresividad a nivel de modelado, que proporciona OOWS 2.0 con respecto a OOWS.

8.2 Trabajo actual y futuro

Actualmente, el método OOWS 2.0 propuesto en esta tesis, continúa en su proceso de validación previo a la implementación de una herramienta industrial de soporte. El trabajo inmediato a desarrollar consiste en la realización del experimento de viabilidad aquí planteado. Mediante el posterior análisis de los datos obtenidos, se espera obtener la retroalimentación necesaria para medir el impacto y la viabilidad de las distintas contribuciones, así como introducir las mejoras que fuesen necesarias. Los resultados proporcionarán una mayor justificación a las contribuciones aquí expuestas.

Paralelamente al desarrollo del experimento, el método OOWS 2.0 se está aplicando, de modo preliminar, en el marco del proyecto Diagen del centro de Investigación ProS. El objetivo principal de este proyecto es el desarrollo de una herramienta informática, para el diagnóstico de enfermedades a partir de la información genética de un paciente. El proyecto plantea el desarrollo de una aplicación Web, que asista al médico o bioinformático en la generación de un informe de diagnóstico. Dado que en esta aplicación aspectos como la usabilidad, la sencillez de utilización y la interacción con el usuario resultan claves, se ha decidido especificar y desarrollar parte de los requisitos, mediante el método OOWS 2.0. Actualmente, este proyecto se encuentra en una fase temprana de desarrollo, en la cual se están especificando los modelos correspondientes y la infraestructura tecnológica que dará soporte a la implementación. Mediante el desarrollo de esta aplicación, se espera

obtener retroalimentación adicional sobre la viabilidad del método, en un entorno de desarrollo real.

A partir del trabajo desarrollado en la presente tesis, surgen distintas líneas de trabajo futuro a tratar:

- **Desarrollo de una herramienta CASE de soporte al método:** el entorno de desarrollo Eclipse ha adquirido una especial relevancia en el ámbito del DSDM. Su principal interés reside en un conjunto de herramientas de soporte para la creación de editores de modelos gráficos, como son los *frameworks* GEF y GMF. Utilizando dichas tecnologías se plantea como trabajo futuro, el desarrollo de un editor gráfico avanzado para la definición de los modelos OOWS 2.0. En este momento, los modelos son creados mediante el editor por defecto que se genera a partir de los distintos metamodelos. Las mejoras que proporcionaría una herramienta visual al método son la simplificación de las tareas de modelado, el incremento de la eficiencia y su difusión académica.

- **Proceso de generación de código:** el método OOWS 2.0 plantea la especificación de una aplicación Web 2.0 como un conjunto de modelos, los cuales son una representación fiel de la misma. La ventaja más interesante de esta aproximación es la posibilidad de generar el código de la aplicación Web 2.0, a partir de dichos modelos, mediante la implementación de un compilador de modelos. La implementación de este tipo de compiladores ya ha sido contemplada en trabajos anteriores. Claros ejemplos son el *Olivanova Transformation Engins* de OO-Method [Pastor & Molina 2007], o el compilador de modelos del método OOWS original [Valverde 2007]. Mediante la implementación de un compilador de modelos es factible la obtención automática de una aplicación Web 2.0 funcional que puede ser probada y utilizada. Como trabajo futuro se plantea la implementación de un compilador de modelos para el método OOWS 2.0. Este compilador será una extensión del compilador previo de OOWS, el cual está implementado mediante un conjunto de transformaciones modelo a código definidas mediante el lenguaje XPand. En este caso particular, se

implementarían las reglas necesarias para la generación de código en una tecnología RIA concreta.

- **Soporte a la integración entre aplicaciones mediante Servicios REST**: la presente tesis se ha centrado, fundamentalmente, en los aspectos de interacción de las aplicaciones Web 2.0. Pero no se debe olvidar otro aspecto muy relevante de estas aplicaciones, como es su alta interoperabilidad. La gran mayoría de aplicaciones Web 2.0 se caracterizan por exponer su funcionalidad, de forma que ésta es fácilmente integrable en otras aplicaciones. Por esta razón, muchas aplicaciones Web 2.0 se construyen en base a la funcionalidad e información proporcionada por otras. Esta exportación de la funcionalidad se realiza, habitualmente en este dominio, mediante los llamados Servicios REST [Richardson & Ruby 2007]. En el marco del método OOWS 2.0, resulta interesante soportar la integración de estos servicios, a nivel de modelado conceptual. De este modo, el analista podría utilizar funcionalidad externa a la hora de modelar su aplicación Web 2.0. Ya existe un primer trabajo en esta línea de investigación [Valverde & Pastor 2009], en el cual se presenta un modelo preliminar con dicho fin. Sin embargo, dicho modelo aún tiene que ser extendido, evaluado e integrado en el método OOWS 2.0

- **Análisis de la aplicación de las contribuciones en otros métodos**: tanto el metamodelo RIA como los patrones Web 2.0 propuestos, han sido definidos sin tener en mente, específicamente, el método OOWS 2.0. Si bien en el marco de esta tesis se introducen en dicho método, la estrategias de integración no están ligadas a un método en concreto. El uso del concepto de *weaving,* en el caso del modelado de la Interfaz RIA, y la utilización de modelos estándar, para la representación conceptual de los patrones Web 2.0, son estrategias suficientemente genéricas para que, en teoría, las contribuciones sean integradas en el marco de otros métodos de Ingeniería Web. No obstante, dicha afirmación tiene que ser evaluada en la práctica, mediante la aplicación de las contribuciones. Como trabajo futuro, se pretende evaluar la integración en otros métodos de Ingeniería Web y, analizar las mejoras o inconvenientes que puedan surgir.

8.3 Publicaciones

Los resultados de la presente tesis han sido publicados en distintos foros académicos relevantes, tanto a nivel nacional como internacional. La Tabla 8.1 muestra un resumen del total de publicaciones realizadas. A continuación, se detallan las publicaciones directamente relacionadas con las contribuciones de la tesis.

Conferencias Nacionales

1. **Valverde, F.,** Panach, I., Aquino, N., Pastor, O.: ***Hacia un Modelo de Interacción Abstracto para la Definición de Interfaces Multiplataforma***. In: Macias, J.A., Granollers, A., Latorre, P. (eds.): VIII Congreso Internacional de Interacción Persona-Ordenador, Interacción 2007 (AIPO). Thomson Editores Spain, Zaragoza, Spain (2007) 251-260

2. **Valverde, F**., Pastor, O.: ***Dealing with REST Service in Model-driven Web Engineering Methods***. In: Acuña, C.J., Castro, V.d., Ruiz-Cortés, A., Pascual, D. (eds.): V Jornadas Científico-Técnicas en Servicios Web y SOA, Madrid, Spain (2009) 243-250

Conferencias Internacionales

3. **Valverde, F.**, Valderas, P., Fons, J.: ***OOWS Suite: Un Entorno de desarrollo para Aplicaciones Web basado en MDA***. In: Losavio, F., Travassos, G.H., Pelechano, V., Díaz, I., Matteo, A. (eds.): IDEAS, X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes Software, Isla Margarita, Venezuela (2007) 253-266

4. Panach, J.I., **Valverde, F**., Pastor, O.: ***Improvement of a Web Engineering Method through Usability Patterns***. In: Weske, M., Hacid, M.S., Godart, C. (eds.): Web Information Systems Engineering – WISE 2007 Workshops, Vol. 4832/2007. Springer (2007) 441-446

5. **Valverde, F**., Panach, J.I., Pastor, O.: ***An Abstract Interaction Model for a MDA Software Production Method***. In: Grundy, J., Hartmann, S., Laender, A.H.F., Maciaszek, L., Roddick, J.F. (eds.): 26th International Conference on Conceptual Modeling (ER 2007), Vol.

83. Australian Computer Society Inc., Auckland, New Zeland (2007) 109-114

6. **Valverde, F.**, Valderas, P., Fons, J., Pastor, O.: *A MDA-Based Environment for Web Applications Development: From Conceptual Models to Code*. In: Brambilla, M., Mendes, E. (eds.): 6th International Workshop on Web-Oriented Software Technologies (IWWOST). Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, Como, Italy (2007) 164-178

7. Panach, J.I., Condori-Fernández, N., **Valverde, F.**, Aquino, N., Pastor, O.: *Early Usability Measurement in Model-Driven Development: Definition and Empirical Evaluation*. In: Elbaum, S., Münch, J. (eds.): 2nd International Symposium on Empirical Software Engineering and Measurement. Association for Computing Machinery, Kaiserlautern, Germany (2008) 354-356

8. **Valverde, F.,** Pastor, O.: *Applying Interaction Patterns: Towards a Model-Driven Approach for Rich Internet Applications Development*. In: Olsina, L., Pastor, O., Schwabe, D., Rossi, G., Winckler, M. (eds.): 7th International Workshop on Web-Oriented Software Technologies, Vol. 445. CEUR-WS, New York, United States (2008) 7-12

9. Vlaanderen, K., **Valverde, F.**, Pastor, O.: *Improvement of a Web Engineering Method Applying Situational Method Engineering*. In: Cordeiro, J., Filipe, J. (eds.): 10th International Conference on Enterprise Information Systems, ICEIS. INSTICC, Barcelona, Spain (2008) 147-154

10. **Valverde, F.**, Pastor, O.: *Facing the Technological Challenges of Web 2.0: A RIA Model-Driven Engineering Approach*. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.): Web Information Systems Engineering - WISE 2009, Vol. 5802. Springer (2009) 131-144

Capítulos de Libro

11. **Valverde, F.,** Panach, I., Aquino, N., Pastor, O.: ***Dealing with Abstract Interaction Modelling in an MDE Development Process: a Pattern-based Approach***. In: Macías, J.A., Granollers, T., Latorre, P. (eds.): New Trends on Human-Computer Interaction. Springer, London (2009) 119-128

12. Panach, J.I., Condori, N., **Valverde, F.** , Aquino, N., Pastor, O.: ***Towards an Early Usability Evaluation for Web Applications.*** In: Cuadrado, J.J., Braungarten, R., Dumke, R., Abran, A. (eds.): Software Process and Product Measurement, Vol. 4895/2008. Springer-Verlag, Berlin Heildeberg (2008) 32-45

13. **Valverde, F.,** Pastor, O., Valderas, P., Pelechano, V.: ***A Model-Driven Engineering Approach for Defining Rich Internet Applications: a Web 2.0 Case Study***. In: Murugesan, S. (ed.): Handbook of Research on Web 2.0, 3.0, and X.0: Technologies, Business, and Social Applications. Information Science Reference (2009) 40-58

14. Vlaanderen, K., **Valverde, F.,** Pastor, O.: ***Model-Driven Web Engineering in the CMS Domain: A Preliminary Research Applying SME***. In: Filipe, J., Cordeiro, J. (eds.): Enterprise Information Systems. Springer (2009) 226-237

Revistas nacionales

15. España, S., Panach, J.I., Aquino, N., **Valverde, F**., Pastor, O.: ***Propuestas para la captura de requisitos y el modelado de la interacción en el marco de MDA***. Novática (2009) 61-67

Tabla 8.1: Publicaciones realizadas en el marco de la tesis.

Tipo	Lugar de publicación	Número
Conferencias Nacionales	Interacción, JSWeb	2
Conferencias Internacionales	IDEAS, IWWOST (2), ESEM, ICEIS, ER, WISE (2)	8
Capítulos de Libro	Springer (3), IGI-Global (1)	4
Revistas Nacionales	Novática	1
	Total	**15**

8.4 Reflexión final

Una de las primeras veces que oí debatir sobre la Web 2.0, fue en una conferencia impartida por uno de los principales acuñadores del término, *Tim O'Reilly*, también conocido por ser el dueño de una de la editoriales más relevantes de libros informáticos. Después de destacar las bondades de la, entonces, nueva filosofía en la Web y otorgarle, casi, el estatus de religión, uno de los asistentes hizo la pregunta clave: ¿Es realmente el término Web 2.0 una forma original de hacer marketing y, poder vender más libros de su editorial?. Obviamente, la pregunta fue recibida con risas por la audiencia, pero era una buena reflexión. Ante semejante pregunta, el conferenciante claudicó y admitió que era cierto que al término le rodeaba cierta aureola de *hype*, palabra anglosajona para denotar algo que recibe una atención desmesurada, de la cual se había aprovechado. No obstante, también recalcó que el término había puesto en el primer plano del desarrollo Web, aspectos clave como la usabilidad, la accesibilidad, la creación colaborativa de contenidos o el rol del usuario final, aspectos que, hasta ese momento, habían quedado relegados por las discusiones meramente tecnológicas. Y que, curiosamente, las ventas de sus libros sobre lenguajes de programación Web se habían resentido desde entonces.

Cuando comencé mi investigación en el ámbito de la Ingeniería Web, el término Web 2.0 era una novedad, una moda. Modas por las que muchas ve-

ces también se guía la investigación, dando un sentido académico a términos atrayentes en la industria del software, tales como *Mobile Web, Cloud Computing, No-SQL* o *Software as a Service*. A pesar de ello, no deja de ser cierto que detrás de estas modas, una vez que se separa la paja del grano, existen contribuciones relevantes que perduran en el tiempo. La motivación de mi investigación era extender el método OOWS para desarrollar aquellas aplicaciones Web, que no era posible especificar mediante sus modelos conceptuales, debido a la evolución imparable de la Web. Y en cierta manera, en el camino, me encontré que esa evolución se englobaba dentro del término Web 2.0. Esta tesis doctoral ha extraído dos conceptos claves en esta filosofía, *Rich Internet Applications* y el énfasis en la interacción con el usuario, y los ha plasmado en el marco del desarrollo de software dirigido por modelos. Ambas contribuciones, más que específicas de la Web 2.0, se han convertido en requisitos estándar de cualquier aplicación Web de calidad. El método OOWS 2.0 es una herramienta más para tratar la complejidad del desarrollo de aplicaciones Web 2.0.

La evolución e implantación de la Web todavía no ha concluido: la Web está más viva que nunca. En los próximos años, la Web se implantará a través de múltiples dispositivos, algunos incluso por definir, en todos los aspectos de nuestra vida diaria. Para lograr esta implantación, surgirán nuevos retos a resolver en el ámbito de la Ingeniería Web. En consecuencia, los métodos de Ingeniería Web tendrán que continuar evolucionando para afrontar estos retos. Independientemente que, por razones de marketing, esta evolución sea denominada como Web 3.0, Web X.0 o Web $n\infty$

Valencia, 6 de octubre de 2010.

Bibliografía

[Alexa 2010] Alexa (2010). The top 500 sites on the web, Alexa Inc., http://www.alexa.com/topsites, Accessed April 2010.

[Alexander 1979] Alexander, C. (1979). *The Timeless Way of Building*. New York, Oxford University Press.

[Aquino, Vanderdonckt *et al.* 2008] Aquino, N., Vanderdonckt, J., Valverde, F. and Pastor, O. (2008). Using Profiles to Support Model Transformations in the Model-Driven Development of User Interfaces. *Computer-Aided Design of User Interfaces VI* V. Lopez Jaquero, F. Montero Simarro, J. P. Molina Masso and J. Vanderdonckt (ed.). London, Springer. **IX:** 35-46.

[Behrens, Clay *et al.* 2009] Behrens, H., Clay, M., Efftinge, S., Eysholdt, M., Friese, P., Kö hnlein, J., Wannheden, K. and Zarnekow, S. (2009). Xtext User Guide 1.0.1**:** 113.

[Bodart, Hennebert *et al.* 1994] Bodart, F., Hennebert, A.-M., Leheureux, J.-M. and Vanderdonckt, J. (1994). Towards a dynamic strategy for computer-aided visual placement. *Proceedings of the workshop on Advanced visual interfaces*. Bari, Italy, ACM Press**:** 78-87.

[Bozzon, Comai *et al.* 2006] Bozzon, A., Comai, S., Fraternali, P. and Carughi, G. T. (2006). Conceptual modeling and code generation

for rich internet applications. *Proceedings of the 6th international conference on Web engineering*. Palo Alto, California, USA, ACM.

[Budinsky, Steinberg *et al.* 2009] Budinsky, F., Steinberg, D., Paternostro, M. and Merks, E. (2009). *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional.

[Bughin & Manyika 2007] Bughin, J. and Manyika, J. (2007). How Businesses are usign Web 2.0: A McKinsey Global Survey

[Calvary, Coutaz *et al.* 2003] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. (2003). "A Unifying Reference Framework for multi-target user interfaces." *Interacting with Computers* **15**(3): 289-308.

[CARE 2009] CARE (2009). Fundamentals of the OLIVANOVA Modeler. Part IV: Presentation Model, Care Technologies S.A.: 206.

[CARE 2010] CARE (2010). OLIVANOVA Tool, CARE Technologies S.A, http://www.care-t.com/products/index.asp, Accessed January 2010.

[Carughi, Comai *et al.* 2007] Carughi, G., Comai, S., Bozzon, A. and Fraternali, P. (2007). Modeling Distributed Events in Data-Intensive Rich Internet Applications. *Web Information Systems Engineering – WISE 2007*: 593-602.

[Ceri, Fraternali *et al.* 2000] Ceri, S., Fraternali, P. and Bongio, A. (2000). "Web Modelling Language (WebML): a modeling language for designing Web sites." *Computer Networks* **33**: 137-157.

[Ceri, Fraternali *et al.* 2003] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S. and Matera, M. (2003). *Designing Data-Intensive Web Applications*. Morgan Kaufmann.

[Cetina, Fons *et al.* 2008] Cetina, C., Fons, J. and Pelechano, V. (2008). Applying Software Product Lines to Build Autonomic Pervasive Systems. *Proceedings of the 2008 12th International Software Product Line Conference*, IEEE Computer Society.

[Chaffer & Swedberg 2009] Chaffer, J. and Swedberg, K. (2009). *Learning jQuery 1.3*. Packt Publishing.

[Clarke, Connors *et al.* 2009] Clarke, J., Connors, J. and Bruno, E. (2009). *JavaFX: Developing Rich Internet Applications*. Prentice Hall.

[Cook & Campbell 1979] Cook, T. and Campbell, D. (1979). *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Wadsworth Publishing.

[Cowan & Lucena 1995] Cowan, D. D. and Lucena, C. J. P. (1995). "Abstract data views: an interface specification concept to enhance design for reuse." *Software Engineering, IEEE Transactions on* **21**(3): 229-243.

[Crumlish & Malone 2009] Crumlish, C. and Malone, E. (2009). *Designing Social Interfaces*. O'Reilly Media, Inc.

[Didonet Del Fabro & Valduriez 2009] Didonet Del Fabro, M. and Valduriez, P. (2009). "Towards the efficient development of model transformations using model weaving and matching transformations." *Software and Systems Modeling* **8**(3): 305-324.

[Dix, Finlay *et al.* 1997] Dix, A., Finlay, J., Abowd, G. and Beale, R. (1997). *Human-computer interaction*. Prentice-Hall, Inc.

[Dolog & Stage 2007] Dolog, P. and Stage, J. (2007). *Designing Interaction Spaces for Rich Internet Applications with UML*. 7th International Conference on Web Engineering, Springer-Verlag.

[Duhl 2003] Duhl, J. (2003). Rich Internet Applicattions - IDC Report.

[Ebay 2010] Ebay (2010). Ebay Desktop Application, Ebay Inc, http://desktop.ebay.com/, Accessed February 2010.

[Escalona 2004] Escalona, M. J. (2004). Modelos y técnicas para la especificación y el análisis de la navegación en sistemas software. *Departamento de Lenguajes y Sistemas Informáticos*. Sevilla, Universidad de Sevilla. **PhD**.

[Fabro, Bézivin *et al.* 2006] Fabro, M. D. d., Bézivin, J. and Valduriez, P. (2006). *Weaving Models with the Eclipse AMW plugin*. Eclipse Summit Europe, Esslingen, Germany.

[Farrell, Lau *et al.* 2009] Farrell, S., Lau, T. and Nusser, S. (2009). Building Communities with People-Tags. *Human-Computer Interaction – INTERACT 2007*: 357-360.

[Fons 2008] Fons, J. (2008). OOWS: Un Mètod Dirigit per Models per al Desenvolupament d'Aplicacions Web. *Departamento de Sistemas Informáticos y Computación*. Valencia, Universidad Politécnica de Valencia. **PhD**.

[Fons, Pelechano *et al.* 2003] Fons, J., Pelechano, V., Albert, M. and Pastor, O. (2003). *Development of Web Applications from Web Enhanced Conceptual Schemas*. ER 2003, LNCS. Springer.

[Fraternali, Tisi *et al.* 2009] Fraternali, P., Tisi, M., Silva, M. and Frattini, L. (2009). Building Community-based Web Applications with a Model-driven Approach and Design Patterns. *Handbook of Research on Web 2.0, 3.0, and X.0: Technologies, Business, and Social Applications*. S. Murugesan (ed.), IGI-Global.

[Gamma, Helm *et al.* 1995] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addyson Wesley.

[Google 2010] Google (2010). Google Web Toolkit, Google Inc., http://code.google.com/intl/webtoolkit/webtoolkit/, Accessed February 2010.

[Hazel 2010] Hazel, P. (2010). PCRE - Perl-compatible regular expressions http://www.pcre.org/pcre.txt, Accessed February 2010.

[Hevner, March *et al.* 2004] Hevner, A. R., March, S. T., Park, J. and Ram, S. (2004). "Design Science in Information Systems Research." *MIS Quarterly* **28**(1): 75-105.

[Hickson & Hyatt 2010] Hickson, I. and Hyatt, D. (2010). HTML5: A vocabulary and associated APIs for HTML and XHTML, W3C, http://dev.w3.org/html5/spec/Overview.html, Accessed April 2010.

[Hitwise 2010] Hitwise (2010). Facebook Reaches Top Ranking in US http://weblogs.hitwise.com/heather-dougherty/2010/03/facebook_reaches_top_ranking_i.html, Accessed March 2010.

[INRIA 2007] INRIA (2007). ATL User Manual 0.7. Nantes, Atlas Group.

[Janssen, Weisbecker *et al.* 1993] Janssen, C., Weisbecker, A. and Ziegler, J. (1993). Generating user interfaces from data models and dialogue net specifications. *Proceedings of the SIGCHI conference on Human factors in computing systems*. Amsterdam, The Netherlands, ACM Press: 418-423.

[Jossic, Del Fabro *et al.* 2007] Jossic, A., Del Fabro, M. D., Lerat, J. P., Bezivin, J. and Jouault, F. (2007). *Model Integration with Model Weaving: a Case Study in System Architecture*. Systems Engineering and Modeling, 2007. ICSEM '07. International Conference on.

[Koch 2006] Koch, N. (2006). Transformation techniques in the model-driven development process of UWE. *Workshop proceedings of the sixth international conference on Web engineering*. Palo Alto, California, ACM.

[Koch, Meliá *et al.* 2008] Koch, N., Meliá , S., Moreno, N., Pelechano, V., Sá nchez, F. and Vara, J. M. (2008). "Model-Driven Web Engineering." *Upgrade* **IX**(2): 40-45.

[Koch, Pigerl *et al.* 2009] Koch, N., Pigerl, M., Zhang, G. and Morozova, T. (2009). *Patterns for the Model-Based Development of RIAs*. Web Engineering.

[Koch. 2000] Koch., N. (2000). Software Engineering for Adaptive Hypermedia Applications. Munich, Ludwig-Maximilians-University of Munich.

[Kühne 2004] Kühne, T. (2004). *What is a Model?* Language Engineering for Model-Driven Software Development, Dagstuhl (Germany).

[Levendovszky, Lengyel *et al.* 2009] Levendovszky, T., Lengyel, L. and Mészáros, T. (2009). "Supporting domain-specific model patterns with metamodeling." *Software and Systems Modeling* **8**(4): 501-520.

[Limbourg & Vanderdonckt 2004] Limbourg, Q. and Vanderdonckt, J. (2004). Usixml: A User Interface Description Language Supporting Multiple Levels Of Independence. *Engineering Advanced Web Applications*. M. Matera and S. Comai (ed.). Paramus, New Jersey, Rinton Press.

[Linaje, Preciado *et al.* 2007a] Linaje, M., Preciado, J. and Sánchez-Figueroa, F. (2007a). A Method for Model Based Design of Rich Internet Application Interactive User Interfaces. *Web Engineering*: 226-241.

[Linaje, Preciado *et al.* 2007b] Linaje, M., Preciado, J. C. and Sánchez-Figueroa, F. (2007b). "Engineering Rich Internet Application User Interfaces over Legacy Web Models." *IEEE Internet Computing*: 53-59.

[Lott, Schall *et al.* 2006] Lott, J., Schall, D. and Peters, K. (2006). *ActionScript 3.0 Cookbook*. O'Reilly Media, Inc.

[March & Smith 1995] March, S. T. and Smith, G. F. (1995). "Design and natural science research on information technology." *Decis. Support Syst.* **15**(4): 251-266.

[Mbaki, Vanderdonckt *et al.* 2008] Mbaki, E., Vanderdonckt, J., Guerrero, J. and Winckler, M. (2008). *Multi-level Dialog Modeling in Highly Interactive Web Interfaces* 7th International Workshop on Web-Oriented Software Technologies, New York, United States, WS-CEUR.

[Meliá & Cachero 2004] Meliá, S. and Cachero, C. (2004). *An MDA Approach for the Development of Web Applications*. 4th International Conference on Web Engineering (ICWE 2004), Munich, Germany, Springer LNCS.

[Melia, Gomez *et al.* 2008] Melia, S., Gomez, J., Perez, S. and Diaz, O. (2008). *A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA*. Web Engineering, 2008. ICWE '08. Eighth International Conference on.

[Mellor, Scott *et al.* 2002] Mellor, S., Scott, K., Uhl, A. and Weise, D. (2002). Model-Driven Architecture. *Advances in Object-Oriented Information Systems*: 233-239.

[Michail 2009] Michail, A. (2009). *Essential Silverlight 3*. Addison-Wesley Professional.

[Molina, Redondo *et al.* 2008] Molina, A. I., Redondo, M. A., Ortega, M. and Hoppe, U. (2008). "CIAM: A Methodology for the Development of Groupware User Interfaces." *Journal of Universal Computer Science* **14**(9): 1435-1446.

[Molina 2003] Molina, P. J. (2003). Especificación de interfaz de usuario: de los requisitos a la generación automática. *Departamento de Sistemas Informáticos y Computación*. Valencia, Universidad Politécnica de Valencia**:** 382.

[Molina, Melia *et al.* 2002] Molina, P. J., Melia, S. and Pastor, O. (2002). *JUST-UI: A User Interface Specification Model.* Proceedings of Computer Aided Design of User Interfaces, CADUI'2002, Valenciennes, Francia.

[Montero, López-Jaquero *et al.* 2005] Montero, F., López-Jaquero, V., Vanderdonckt, J., González, P. and Lozano, M. (2005). *Solving the mapping problem in user interface design by seamless integration in IdealXML*. Proc. of DSV-IS'05, Newcastle upon Tyne, United Kingdom, Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[Mori, Paterno *et al.* 2004] Mori, G., Paterno, F. and Santoro, C. (2004). "Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions." *IEEE Transactions on Software Engineering*.

[Mullet 2003] Mullet, K. (2003). The Essence of Effective Rich Applications, Macromedia Inc.

[Murugesan 2007] Murugesan, S. (2007). "Understanding Web 2.0." *IT Professional* **9**(4): 34-41.

[Murugesan, Deshpande *et al.* 2001] Murugesan, S., Deshpande, Y., Hansen, S. and Ginige, A. (2001). Web Engineering: a New Discipline for Development of Web-Based Systems. *Web Engineering***:** 3-13.

[Nóbrega, Nunes *et al.* 2006] Nóbrega, L., Nunes, N. and Coelho, H. (2006). Mapping ConcurTaskTrees into UML 2.0. *Interactive Systems***:** 237-248.

[Noda & Helwig 2005] Noda, T. and Helwig, S. (2005). Rich Internet Applications - Technical Comparison and Case Studies of AJAX, Flash, and Java based RIA www.uwebc.org/opinionpapers/archives/docs/RIA.pdf, Accessed October 2010.

[Nunes & Cunha 2001] Nunes, N. J. and Cunha, J. F. e. (2001). Wisdom-Whitewater interactive system development with object models. *Object modeling and user interface design: designing interactive systems*, Addison-Wesley Longman Publishing Co., Inc: 197-243.

[O'Reilly 2005] O'Reilly, T. (2005). What is Web 2.0?. Design Patterns and Business Models for the Next Generation of Software http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, Accessed October 2010.

[OAW 2008] OAW (2008). openArchitectureWare 4.3 framework http://www.openarchitectureware.org/, Accessed September 2008.

[OMG 2006] OMG (2006). Meta Object Facility (MOF) Core Specification v2.0, Object Management Group: 88.

[OMG 2008] OMG (2008). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification 1.0, Object Management Group: 240.

[Panach, Condori *et al.* 2008] Panach, J. I., Condori, N., Valverde, F., Aquino, N. and Pastor, O. (2008). Towards an Early Usability Evaluation for Web Applications. *Software Process and Product Measurement*. J. J. Cuadrado, R. Braungarten, R. Dumke and A. Abran (ed.). Berlin Heildeberg, Springer-Verlag. **4895/2008:** 32-45.

[Pastor & Molina 2007] Pastor, O. and Molina, J. C. (2007). *Model-Driven Architecture in Practice. A Software Production Environment Based on Conceptual Modelling*. Berlin Heildeberg, Springer-Verlag.

[Paternò 2004] Paternò, F. (2004). ConcurTaskTrees: An Engineered Notation for Task Models. *The Handbook of Task Analysis for*

Human-Computer Interaction. D. Diaper, N. Stanton and N. A. Stanton (ed.). London, United Kingdom, Lawrence Erlbaum Associates: 483-501.

[Paternò, Mancini *et al.* 1997] Paternò, F., Mancini, C. and Meniconi, S. (1997). ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. *Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, Chapman & Hall, Ltd.: 362-369.

[Pérez, Díaz *et al.* 2008] Pérez, S., Díaz, O., Meliá, S. and Gómez, J. (2008). *Facing Interaction-Rich RIAs: the Orchestration Model*. Eight International Conference on Web Engineering, New York, USA, IEEE Computer Society.

[Preciado, Linaje *et al.* 2008] Preciado, J. C., Linaje, M., Morales-Chaparro, R., Sanchez-Figueroa, F., Zhang, G., Kroi, C. and Koch, N. (2008). Designing Rich Internet Applications Combining UWE and RUX-Method. *Proceedings of the 2008 Eighth International Conference on Web Engineering - Volume 00*, IEEE Computer Society.

[Preciado, Linaje *et al.* 2005] Preciado, J. C., Linaje, M., Sánchez, F. and Comai, S. (2005). *Necessity of methodologies to model Rich Internet Applications*. 7th IEEE International Symposium on Web Site Evolution, Budapest, Hungary, IEEE.

[Puerta & Maulsby 1997] Puerta, A. and Maulsby, D. (1997). *MOBI-D: A Model-Based Development Environment for User Centered Design*. Proceedings of CHI'97, Atlanta, EEUU, ACM Press.

[Raggett, Hors *et al.* 1999] Raggett, D., Hors, A. L. and Jacobs, I. (1999). HTML 4.01 Specification, W3C.

[Richardson & Ruby 2007] Richardson, L. and Ruby, S. (2007). *RESTful Web Services*. O'Reilly Media, Inc.

[Rossi, Pastor *et al.* 2008] Rossi, G., Pastor, O., Schwabe, D. and Olsina, L., Eds. (2008). *Web Engineering: Modelling and Implementing Web Applications*. Human-Computer Interaction Series, Springer.

[Rossi 2000] Rossi, G., Schwabe, D., Lyardet, F. (2000). *User interface patterns for hypermedia applications*. Proceedings of the working conference on Advanced visual interfaces, Palermo, Italy, ACM Press.

[Rossi, Urbieta *et al.* 2008] Rossi, G., Urbieta, M., Ginzburg, J., Distante, D. and Garrido, A. (2008). *Refactoring to Rich Internet Applications: A Model-Driven Approach*. Eight International Conference on Web Engineering, New York, United States, IEEE Computer Society.

[Ruiz 2007] Ruiz, F. J. M. (2007). A Development Method for User Interfaces of Rich Internet Applications. Louvain, Belgium, Université catholique de Louvain**:** 95.

[Saeki 2003] Saeki, M. (2003). Embedding metrics into information systems development methods: an application of method engineering technique. *Proceedings of the 15th international conference on Advanced information systems engineering*. Klagenfurt, Austria, Springer-Verlag.

[Schwabe, Rossi *et al.* 1996] Schwabe, D., Rossi, G. and Barbosa, S. (1996). *Systematic Hypermedia Design with OOHDM*. ACM Conference on Hypertext, Washington, USA.

[Schwabe 1998] Schwabe, D., Rossi, G. (1998). An object oriented approach to Web-based applications design. *Theory and Practice of Object Systems*, John Wiley & Sons, Inc. **4:** 207 - 225.

[Sencha 2010] Sencha (2010). EXT JS: Cross-Browser Rich Internet Application Framework, Sencha Inc., http://www.sencha.com/products/js/, Accessed October 2010.

[SIIA 2008] SIIA (2008). Business Use of Web 2.0, Software & Industry Software Association, http://www.siia.net/content/pubs/web2survey_0108.pdf, Accessed October 2008.

[Silva & Paton 2003] Silva, P. P. d. and Paton, N. W. (2003). "User Interface Modeling in UMLi." *IEEE Software* **20**(4): 62-69.

[Swithinbank, Chessell *et al.* 2005] Swithinbank, P., Chessell, M., Gardner, T., Griffin, C., Man, J., Wylie, H. and Yusuf, L. (2005). Patterns: Model-Driven Development Using IBM Rational Software Architect, IBM Redbooks.

[Tapper, Labriola *et al.* 2008] Tapper, J., Labriola, M., Boles, M. and Talbot, J. (2008). *Adobe Flex 3: training from the source*. Adobe Press.

[Tidwell 2005] Tidwell, J. (2005). *Designing Interfaces*. O'Reilly Media.

[Toxboe 2009] Toxboe, A. (2009). UI Patterns www.ui-patterns.com, Accessed October 2010.

[Traetteberg, Molina *et al.* 2004] Traetteberg, H., Molina, P. J. and Nunes, N. J. (2004). Making model-based UI design practical: usable and open methods and tools. *Proceedings of the 9th international conference on Intelligent user interface*. Funchal, Madeira, Portugal, ACM Press: 376-377.

[Travassos, Fabbri *et al.* 2005] Travassos, G. H., Fabbri, S. and Maldonado, J. C. (2005). Experimental Software Engineering: An Introduction - Tutorial. *19th Brazilian Symposium on Software Engineering (SBES 2005)*. Uberlandia / MG, Brazil.

[Troyer, Casteleyn *et al.* 2008] Troyer, O. d., Casteleyn, S. and Plessers, P. (2008). WSDM: Web Semantics Design Method. *Web Engineering: Modelling and Implementing Web Applications*, Springer: 303-352.

[Urbieta, Rossi *et al.* 2007] Urbieta, M., Rossi, G., Ginzburg, J. and Schwabe, D. (2007). Designing the Interface of Rich Internet Applications. *Fifth Latin American Web Congress (LA-WEB)*. Santiago de Chile.

[Valderas 2008] Valderas, P. (2008). A Requirements Engineering Approach for the Development of Web Applications. *Information Systems and Computation*. Valencia, Universidad Politécnica de Valencia. **PhD**.

[Valderas, Fons *et al.* 2005] Valderas, P., Fons, J. and Pelechano, V. (2005). *Transforming Web Requirements into Navigational Models: AN MDA Based Approach.* 24th International Conference on Conceptual Modeling (ER 2005), Klagenfurt, Austria.

[Valverde 2007] Valverde, F. (2007). Design and Implementation of an MDA Environment for Web applications development. *Department of Information Systems and Computation*. Valencia, Universidad Politécnica de Valencia**:** 126.

[Valverde & Pastor 2009] Valverde, F. and Pastor, O. (2009). *Dealing with REST Service in Model-driven Web Engineering Methods*. V Jornadas Científico-Técnicas en Servicios Web y SOA, Madrid, Spain.

[Valverde, Valderas *et al.* 2007] Valverde, F., Valderas, P., Fons, J. and Pastor, O. (2007). *A MDA-Based Environment for Web Applications Development: From Conceptual Models to Code*. 6th International Workshop on Web-Oriented Software Technologies (IWWOST), Como, Italy, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy.

[Van Wellie 2000] Van Wellie, M., Traetteberg, H. (2000). *Interaction Patterns in User Interfaces*. PLoP 2000, Allerton Park Monticello, Illinois, USA.

[Vanderdonckt, Limbourg *et al.* 2004] Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D. and Florins, M. (2004). *USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces*. Proceedings of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, Greece.

[Weerd & Brinkkemper 2009] Weerd, I. v. d. and Brinkkemper, S. (2009). Meta-Modeling for Situational Analysis and Design Methods. *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications*, IGI-Global: 35-54.

[Wieringa 2008] Wieringa, R. (2008). Requirements Engineering Research Methodology: Principles and practice, University of Twente.

[Wohlin, Runeson *et al.* 2000] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. and Wesslén, A. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers.

[Wright & Dietrich 2008] Wright, J. M. and Dietrich, J. B. (2008). Requirements for Rich Internet Application Design Methodologies. *Proceedings of the 9th international conference on Web Information Systems Engineering*. Auckland, New Zealand, Springer-Verlag.

[Yahoo 2008] Yahoo (2008). Yahoo Design Pattern Library, Yahoo! Inc., http://developer.yahoo.com/ypatterns/, Accessed October 2010.

[Zelkowitz & Wallace 1997] Zelkowitz, M. V. and Wallace, D. (1997). "Experimental validation in software engineering." *Information and Software Technology* **39**(11): 735-743.

Anexos

A.1 Ejemplo ilustrativo: *My Social Research*

El objetivo principal del ejemplo ilustrativo planteado es el de gestionar la actividad científica de un investigador y, la creación de comunidades alrededor de diversos temas de investigación científica. En otras palabras, se pretende trasladar el concepto de Web Social al entorno académico, tal y como *Facebook* (www.facebook.com) ha hecho en el entorno personal o *LinkedIn* (www.linkedin.com) en el entorno profesional. Las ventajas potenciales de una aplicación Web de estas características son notables. Sin lugar a dudas, la labor investigadora implica estar al corriente de los nuevos trabajos en el área, así como establecer relaciones con otros investigadores.

En la actualidad, los mecanismos disponibles en la Web para la recopilación de los trabajos relevantes en un área científica son los buscadores académicos y las páginas personales. Los buscadores académicos se centran en indexar las distintas conferencias y revistas, proporcionando resultados más adecuados que los buscadores tradicionales. Ejemplos son *Google Scholar* (scholar.google.com) o, buscadores de las propias editoriales como *SpringerLink* (www.springerlink.com) o *IEEEXplore* (ieeexplore.ieee.org). Por otra parte, muchos investigadores deciden crear una página Web para hacer público su trabajo. Aunque este último método es una buena opción, habitualmente dichas páginas personales son muy simples o se encuentran obsoletas, ya que mantenerlas continuamente actualizadas supone un gran esfuerzo. En consecuencia, una aplicación Web 2.0 para la difusión sencilla de la investigación científica y el descubrimiento de nuevos trabajos relevantes, proporciona unos beneficios tangibles.

Esta problemática no ha pasado desapercibida y ya es posible encontrar diversas Web Sociales con un claro enfoque académico. Algunos ejemplos son *Academia* (www.academia.edu), *CiteULike* (www.citeulike.org) o *Epernicus* (www.epernicus.com). Sin embargo, a diferencia de otros entornos, actualmente ninguna ha adquirido una relevancia suficiente como para ser usada masivamente por la comunidad científica. La aplicación Web 2.0 que se propone, "*My Social Research*", recopila la funcionalidad social más destacada de estas Webs, a fin de utilizar un conjunto de escenarios reales, que sirvan para ilustrar los modelos introducidos en la presente tesis doctoral. Cabe señalar que la descripción del ejemplo ilustrativo no pretende ser una especificación de requisitos exhaustiva. Por lo tanto, se han obviado funcionalidades genéricas y habituales para centrarse, esencialmente, en la funcionalidad que proporciona la perspectiva social a la aplicación. A continuación, se realiza una breve descripción de los requisitos en los cuales se basa nuestro ejemplo:

Gestión del perfil del investigador: para utilizar la aplicación los distintos usuarios tienen que crear su perfil investigador. Dicho perfil está formado por los datos personales básicos (nombre, dirección de correo, nacionalidad) además de la información académica. Seguidamente, el usuario tiene que introducir su currículum académico, el cual incluya sus publicaciones relevantes, participaciones en congresos, actividad docente, etc. El perfil tiene que contener la afiliación universitaria, así como un conjunto de líneas de investigación en la cuales el investigador tiene experiencia y, opcionalmente, aquéllas de su interés. A partir de la información introducida, el sistema le sugiere, automáticamente, su inclusión en redes de investigación relacionadas y la subscripción a notificaciones relevantes.

Mi Librería: la librería contiene los resultados de investigación que el usuario decida difundir. Todo resultado se especifica con una descripción breve del mismo y un fichero disponible para el resto de los usuarios. Estos resultados se clasifican en artículos (documentos enviados a revistas o congresos), presentaciones (ponencias, tutoriales, etc.), ficheros (por ejemplo código fuente, herramientas, etc.) y demos o *screencasts* (videos sobre simulaciones o utilización de herramientas). Cada elemento de la librería es etiquetado, mediante palabras clave acordes a su contenido, para facilitar las búsquedas posteriores. La librería también incorpora un mecanismo, con el propósito que los visitantes evalúen la calidad de los distintos contenidos y

proporcionen retroalimentación al usuario. También, se proporciona un mecanismo de privacidad para decidir qué elementos son públicos o cuáles son visibles, para una red determinada.

Mis redes de investigación: el usuario puede pertenecer a varias redes de investigadores las cuales se crean, o bien en función de la institución académica o localización geográfica del investigador, o bien en base a los distintos temas de investigación definidos en la aplicación. Una vez que un usuario pertenezca a una red, la información que actualice en su librería es difundida a través de la misma. De la misma forma, el usuario recibirá aquellos cambios que han producido los diferentes miembros de la red, como nuevas incorporaciones de material a las distintas librerías o el anuncio de eventos relevantes. Esta información es clasificada en función de su interés, fecha y tipo, siendo mostrada al usuario cuando inicia sesión en la aplicación. También, de dispone de la creación de redes privadas formadas por investigadores, que hayan sido invitados a las mismas.

Mis favoritos: los usuarios pueden crear una colección de favoritos, basada en una recopilación de materiales científicos o de contactos de investigadores. En primer lugar, se puede recomendar material científico, de diversa índole, como artículos, presentaciones, etc. que el usuario considere de especial interés para la comunidad. Con dicho fin, el usuario proporcionará una descripción y una evaluación que determine el grado de interés. Este material recomendado puede provenir desde fuentes externas de la aplicación. Respecto a los investigadores, el usuario selecciona aquellos que considere relevantes, de tal forma que se produce una suscripción directa a los cambios que éstos realicen en su perfil o librería. Mediante este mecanismo, se obtiene, indirectamente, la popularidad de un investigador determinado y, el propio usuario puede descubrir a otros investigadores que están interesados en su trabajo.

Calendario: otro de los objetivos de la aplicación es que el usuario pueda mantener un calendario con sus eventos científicos. En este calendario, el usuario incluye las fechas que él considere relevantes para su investigación. Una de las características más interesantes de esta funcionalidad, es que además se agregan, automáticamente, las fechas límites de envío de trabajos y de notificación de aceptación, pertenecientes a conferencias o revistas rela-

cionadas con su investigación. De esta manera, el usuario recibe información relevante para la publicación de sus resultados y, puede ajustar a su planificación.

Terminología colaborativa: en el ámbito científico es una tarea fundamental consensuar una terminología, en el marco de una disciplina, para que los distintos trabajos sean concisos a la hora de presentar su contribución. A pesar de todo, es habitual que en una disciplina concreta, un concepto no sea utilizado con exactamente el mismo significado, por distintos autores. Esta situación ocurre, muchas veces, por el desconocimiento de un trabajo que había introducido el término previamente. Este hecho provoca confusión y ambigüedad puesto que, dependiendo de las referencias utilizadas, varía significativamente el sentido del trabajo. Aprovechando los beneficios de las redes sociales, la aplicación introduce un mecanismo similar a los populares *wikis*, con el fin que sea la propia comunidad quién decida qué definición es la más adecuada para un término. Esta edición colaborativa de la definición tiene que estar respaldada por las referencias bibliográficas necesarias y, además, es evaluada por la propia comunidad. En consecuencia, entre las distintas definiciones propuestas, será posible conocer cual es la que posee una mayor aceptación y en base a qué referencias bibliográficas.

Pagina Personal: además de poder gestionar la información académica, es interesante que usuarios externos, que no estén registrados en la aplicación, accedan a la información sobre los investigadores. Por lo tanto, se le proporciona al usuario un mecanismo para la creación sencilla de una página Web personal académica. Esta página pública se genera, a partir del material científico y de la información sobre su perfil, que se encuentre disponible en la aplicación. Esta funcionalidad simplifica la tarea de crear este tipo de páginas Web personales y, permite que los contenidos sean indexados por los buscadores Web.

A.2 Metamodelo de Interacción Abstracto

Este anexo describe las diferentes vistas que forman el metamodelo de interacción abstracto. El metamodelo se ha definido utilizando EMF (*Eclipse Modelling Framework*), un *framework* que utiliza como lenguaje de metamodelado *Ecore*, un subconjunto del lenguaje MOF. En esta sección, se describe brevemente la estructura del metamodelo. La semántica específica de cada entidad es abordada con detalle en la sección 4.3. Cabe destacar que algunas primitivas del Modelo de Interacción Abstracto se crean de forma textual. Para especificarlas en el metamodelo se han incluido las entidades de metamodelado generadas, automáticamente, a partir de la gramática XText especificada. Estas entidades se han agrupado en el paquete del metamodelo *syntax*.

La Fig. A.1 muestra la vista principal del metamodelo, el cual se define a partir de la entidad raíz *Abstract Interaction Model*. Con esta entidad se relacionan las distintas entidades que representan los conceptos fundamentales del metamodelo, como son los usuarios, las *AIU* y los *Interaction Contexts*. Mediante las relaciones correspondientes, se representan los conceptos del Diagrama de Usuarios y del Mapa de Interacción. Toda entidad del metamodelo es una especialización de la entidad *NamedElement*, compuesta de un nombre y un alias. Ambas propiedades identifican las primitivas conceptuales a la hora de construir el modelo.

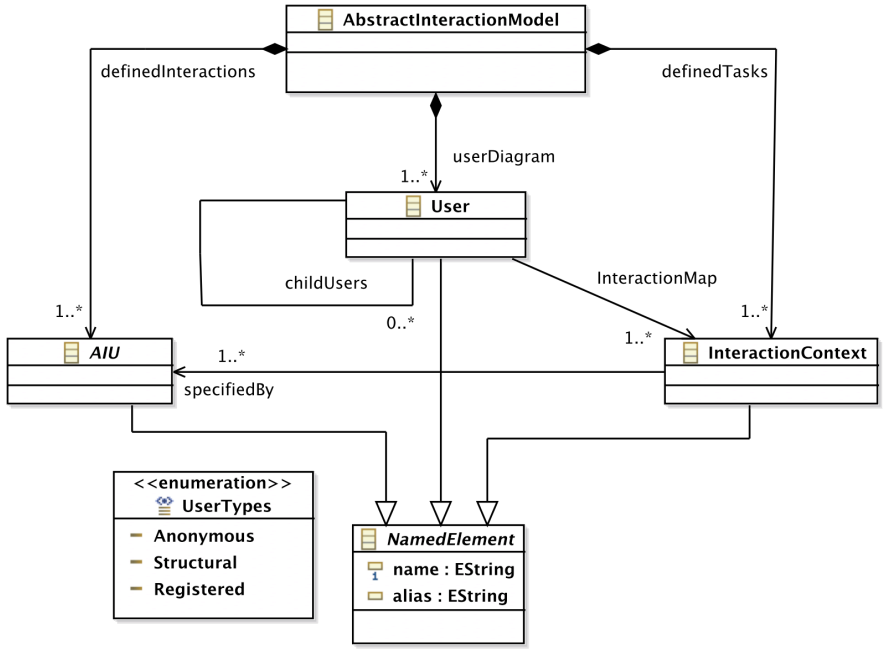


Fig. A.1: Vista Principal del Metamodelo RIA

La Fig. A.2 muestra la vista para la definición de la *Population AIU*. Esta entidad se compone de una *Population Class*, definida como directora a través de la relación *manager*, un conjunto opcional de *Population Classes*, que define las clases complementarias, y un conjunto de *Population Relationships*, que define las relaciones especificadas en la AIU entre estas clases. Al tratarse de una vista sobre el Modelo de Objetos de OO-Method, se incluyen las relaciones pertinentes con las entidades de dicho modelo.

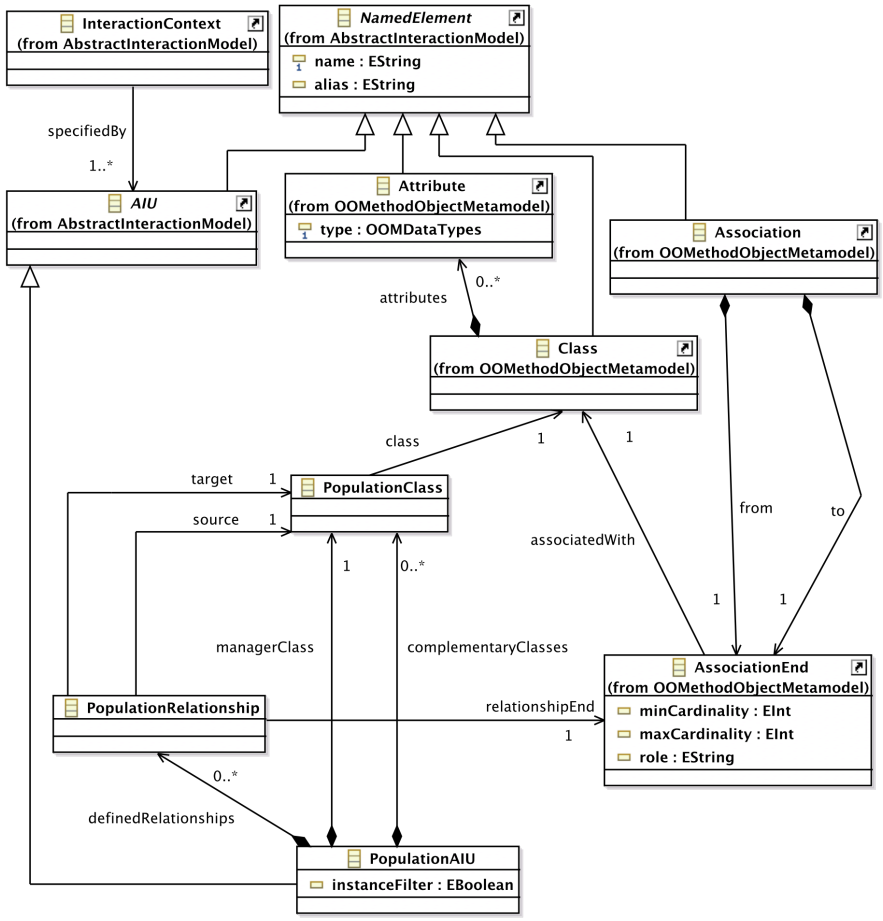


Fig. A.2: Vista de la *Population AIU*

La Fig. A.3 muestra la vista que define la *Service AIU*. Esta entidad se compone de una relación con un servicio del Modelo de Objetos OO-Method, y un conjunto de entidades *Argument View*, que hacen referencia a los argumentos del servicio que son utilizados en la AIU. Los Patrones Auxiliares de Interacción se definen como una especialización de la entidad *AIP*. Un ejemplo es el patrón *Argument Setting* que se muestra en la Fig. A.4. Este patrón se encuentra ligado a la especificación de una *Service AIU*. La entidad que representa el patrón, se relaciona con un *Argument View* para especificar la regla de inicialización del valor del argumento correspondiente. La propie-

dad *st_rule* de la entidad *Argument Setting* hace referencia a la definición de una regla de inicialización estática mientras que la entidad *Dynamic rule*, describe las propiedades necesarias para la definición de reglas de inicialización dinámicas.

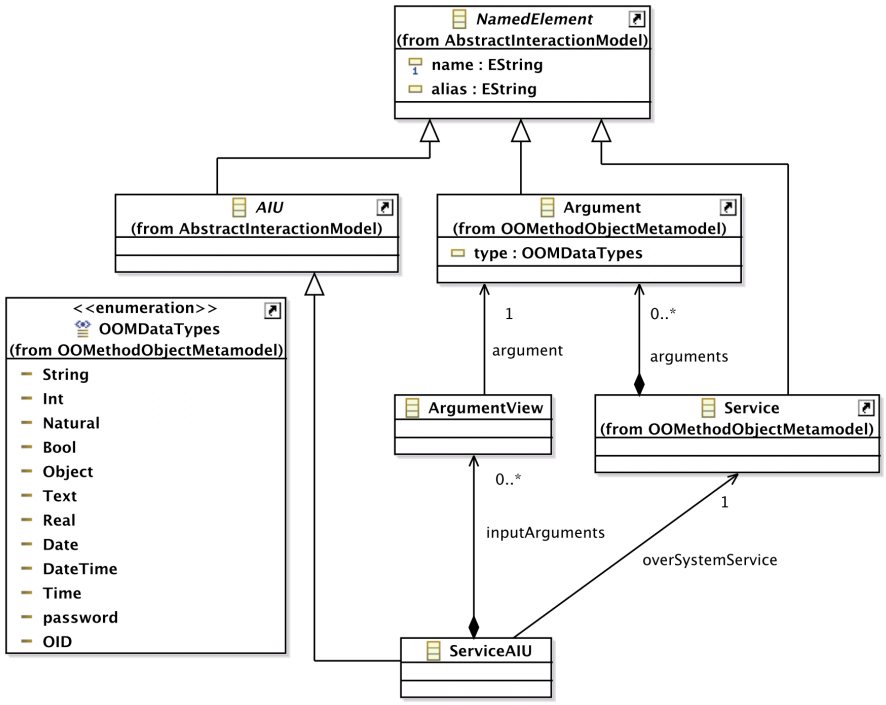


Fig. A.3: Vista de la *Service AIU*

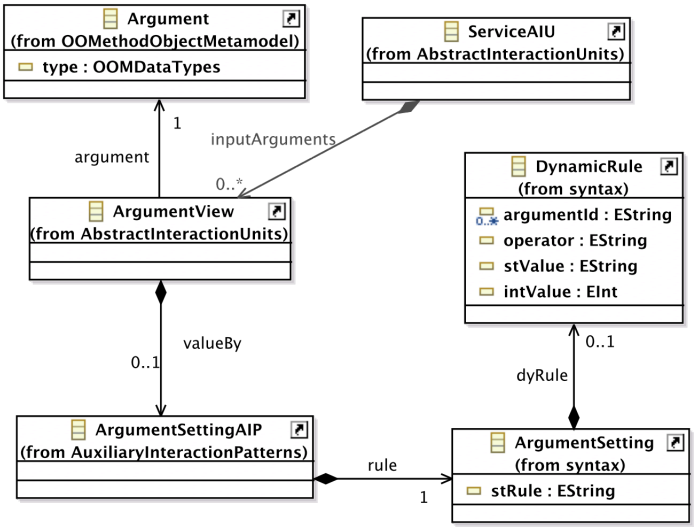


Fig. A.4: Vista del *Argument Setting AIP*

La Fig. A.5 muestra la representación de dos AIP asociados a la *Population AIU*: *Pagination* y *Order Criteria*. El primer patrón únicamente se define mediante un conjunto de propiedades que configuran la paginación de la AIU. Por otra parte, el patrón *Order Criteria* se define mediante un conjunto de formulas textuales representadas por las entidades *Order Criteria* y *Attribute Criteria*. Cada *Attribute Criteria* representa un atributo que participa en el criterio de ordenación en un sentido determinado (Ascendente o Descendente).

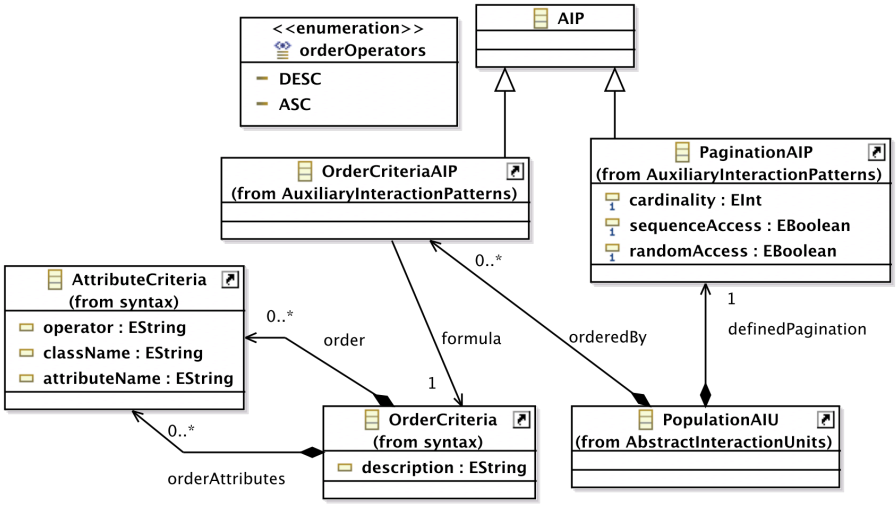


Fig. A.5: Vista de los AIP *Pagination* y *Order Criteria*

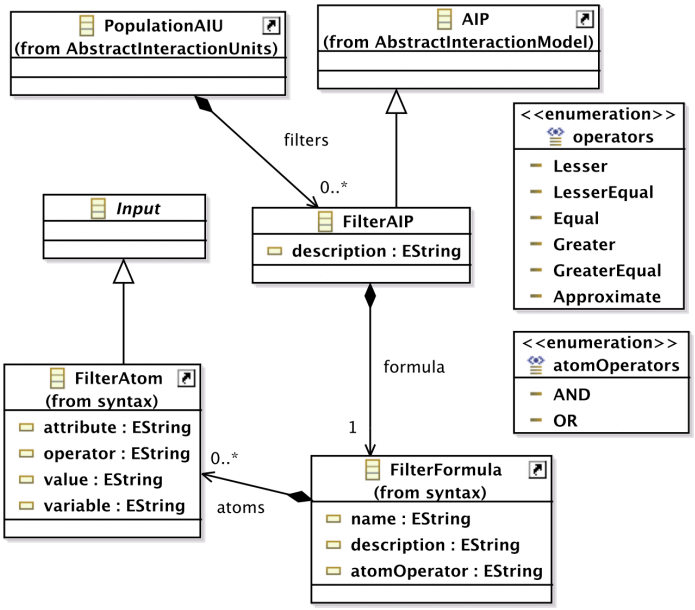


Fig. A.6: Vista del *Filter AIP*

El patrón *Filter* se muestra en la Fig. A.6 asociado también a la *Population AIU*. Los filtros se definen mediante una fórmula textual, compuesta de varios átomos concatenados (entidad *FilterAtom*) a través de un *atomOperator*. Cada átomo se define mediante un atributo, un valor o una variable, y un operador de comparación (enumeración *operators*).

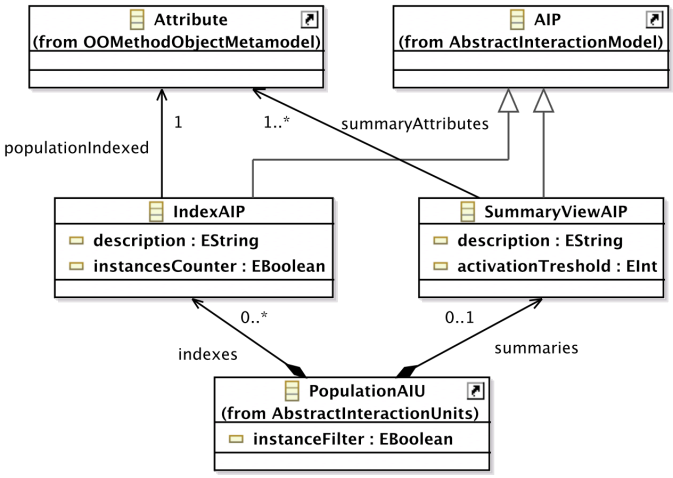


Fig. A.7: Vista de los AIP *Summary View* y *Index*

La Fig. A.7 muestra los dos AIP utilizados para resumir la información de una *Population AIU*: *Summary View* y *Index*. Ambos patrones se asocian a la entidad *Attribute*, mediante una relación única en el caso del índice, o mediante una relación múltiple en el caso de la vista resumida. Además de la descripción, el *Index AIP* incluye la propiedad *instancesCounter* para mostrar el número de instancias de cada categoría del índice. Por otra parte, el patrón *Summary View AIP* incluye la propiedad *activationThreshold*, que determina el número de instancias a partir del cual se activa, automáticamente, la vista resumida.

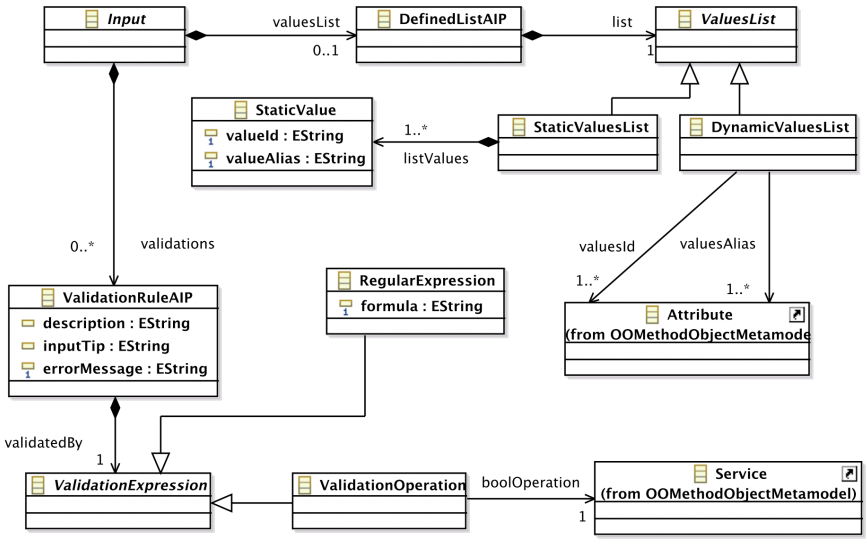


Fig. A.8: Vista de los AIP *Defined List* y *Validation Rule*

La Fig. A.8 muestra la vista de los AIP *Defined List* y *Validation Rule*. Ambos patrones se asocian a un elemento del metamodelo especializado a partir de *Input*: *Argument View* o *Filter Atom*. El *Defined List AIP* se compone de una lista de valores que puede ser estática o dinámica. Una lista estática se define a partir de un conjunto de *Static Values* descritos mediante un valor y un identificador. Por otro lado, una lista dinámica se define mediante dos relaciones con la entidad *Attribute*: una para definir los identificadores (*valuesId*) y, otra para obtener los valores de la lista (*valuesAlias*). El *Validation Rule AIP* se define en función de una *Validation Expression*. Esta expresión se define, o bien como una expresión regular (entidad *RegularExpression*), o bien mediante una *Validation Operation* asociada con un servicio del Modelo de Objetos OO-Method.

El patrón auxiliar *Instance Edition* se muestra en la Fig. A.9. El patrón se relaciona con una *Population AIU* que representa las instancias a modificar. Además, se compone de tres relaciones opcionales con la entidad *Service AIU* para definir las operaciones de modificación de las instancias. La entidad *Association Rule* describe, textualmente, las reglas para inicializar el valor de los argumentos de los servicios asociados al patrón.

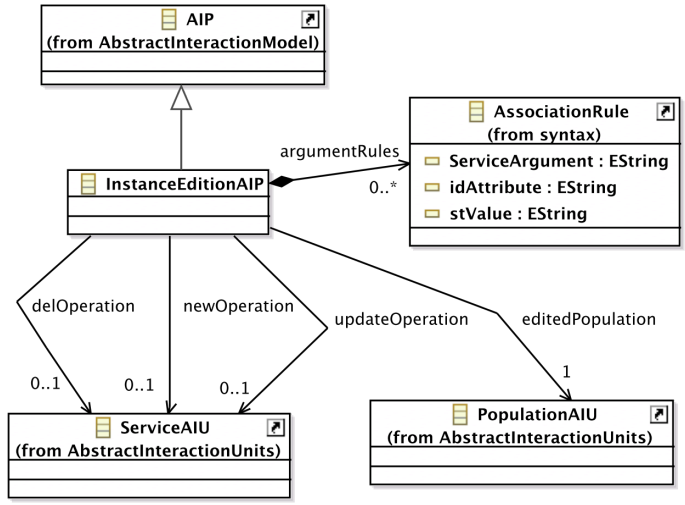


Fig. A.9: Vista del *Instance Edition AIP*

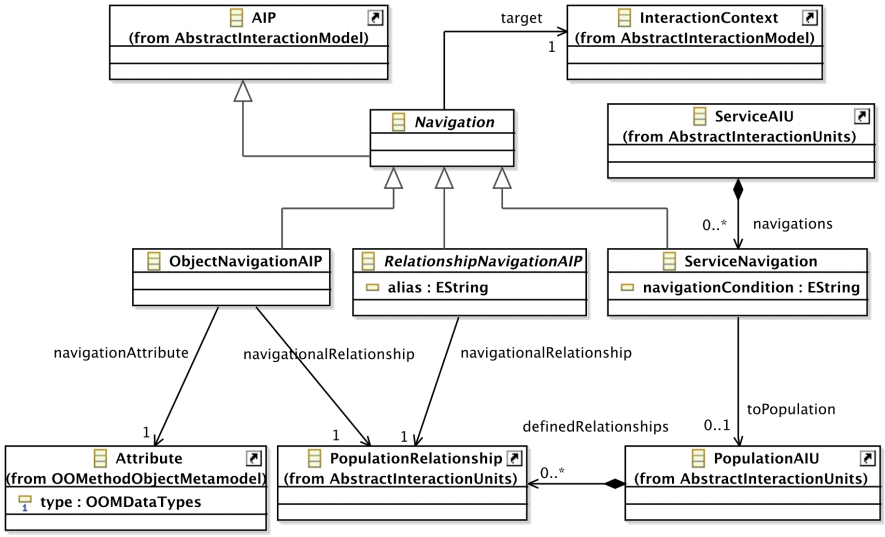


Fig. A.10: Vista de los AIP de Navegación

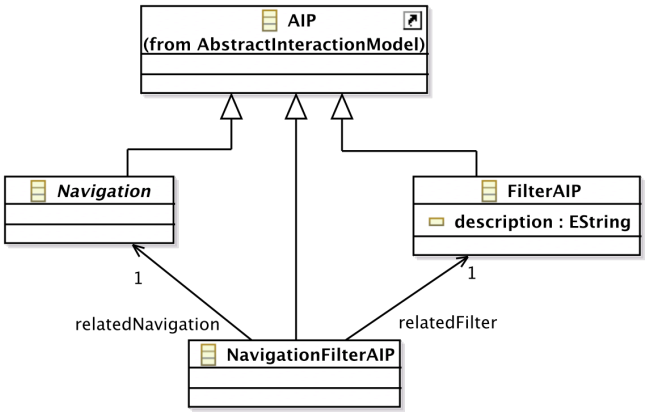


Fig. A.11: Vista del *Navigation Filter AIP*

La Fig. A.10 muestra las distintas entidades que representan la navegación: *Object Navigation, Relationship Navigation y Service Navigation*. Estas tres entidades se especializan a partir de la entidad *Navigation,* y se relacionan con el *Interaction Context* destino de la navegación (relación *target*). Tanto el *Object Navigation AIP* como la *Relationship Navigation AIP*, se relacionan con una *Population Relationship*, que enlaza la clase que origina la navegación con la clase que muestra la información en el *Interaction Context* destino. El patrón *Object Navigation AIP* incluye una relación con un atributo de la *Population AIU*. Por otro lado, el *Service Navigation AIP,* se especifica a partir de una *Service AIU* y se relaciona con la *Population AIU* que es mostrada una vez finalizado el servicio. La entidad incluye la propiedad *navigationCondition* para establecer una condición que se tiene que cumplir para activar la navegación. Por último, la Fig. A.11 muestra el *Navigation Filter AIP*. Este AIP se especifica mediante una relación *relatedNavigation* con una navegación, y una relación *relatedFilter* con un filtro a aplicar.

A.3 Metamodelo RIA para Adobe Flex

El metamodelo RIA para *Adobe Flex* es una especialización a partir del metamodelo introducido en la sección 5.2. Dicho metamodelo, que se muestra en la Fig. A.12, define las entidades abstractas a través de las cuales se especializa un metamodelo tecnológico específico. Este metamodelo se define en el paquete *core,* mientras que el resto de entidades del metamodelo para *Adobe Flex* se definen en el paquete *flex*. Las distintas vistas del metamodelo se muestran desde la Fig. A.13 a la Fig. A.17. Cada vista representa un conjunto de *widgets* de *Adobe Flex*, que guardan en común la función interactiva que desempeñan. Por ejemplo, la Fig. A.14 representa los *widgets* para la visualización de información.

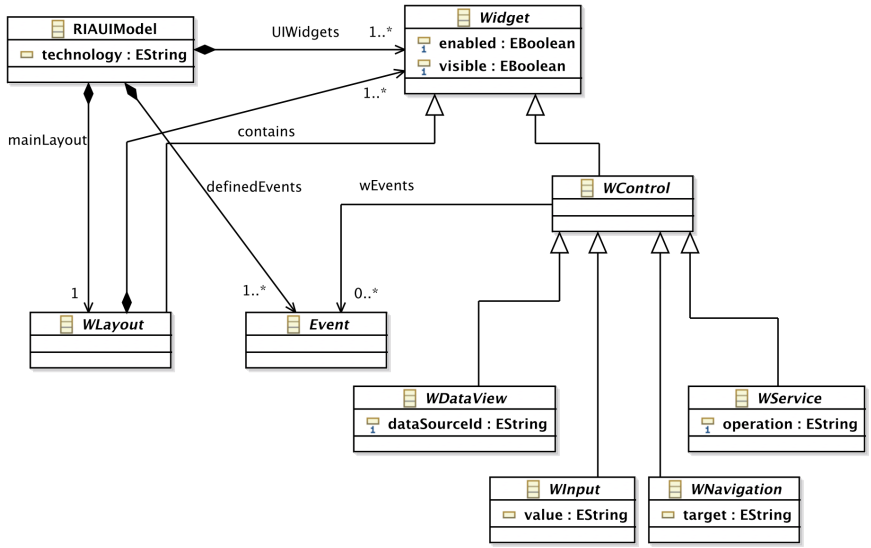


Fig. A.12: Vista *Core* del Metamodelo RIA

Todo *widget* es introducido en el metamodelo mediante una entidad con el mismo nombre encabezada por el prefijo "F". Cada *widget* incluye un conjunto de propiedades (atributos de la entidad), que guardan una relación directa con aquéllas presentes en el lenguaje de programación *ActionScript*. Únicamente, se han representado las propiedades que presentan algún tipo de interés desde el punto de vista de la interacción. Aquellas propiedades que

modifican las propiedades visuales de los *widgets* han sido omitidas. Adicionalmente, se ha definido un *widget* abstracto *FCommonWidget* (ver Fig. A.13). Este *widget* no está presente en *Adobe Flex*, pero se ha utilizado para definir un conjunto de propiedades que son comunes a todos los *widgets*. Por lo tanto, todas las entidades del metamodelo son especializadas a partir de este *widget*. Por último cabe destacar, que el metamodelo no es una representación completa de todos los *widgets* presentes en la tecnología *Adobe Flex*. Dada la gran variedad de *widgets* que proporciona, tan solo se han especificado aquéllos utilizados en el contexto de la tesis doctoral.

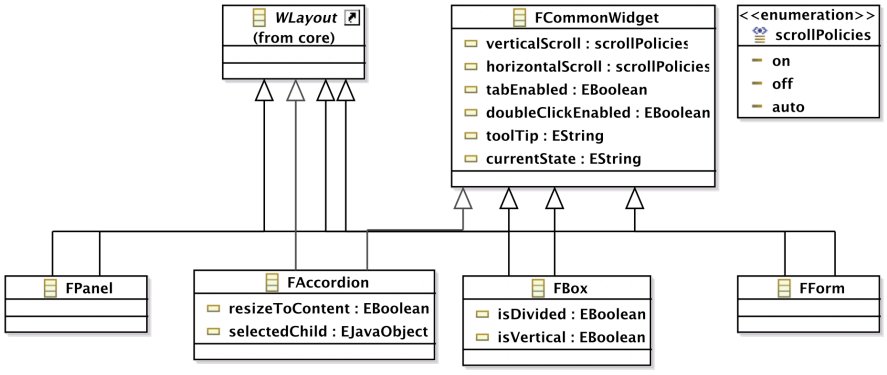


Fig. A.13: Entidades *Layout*

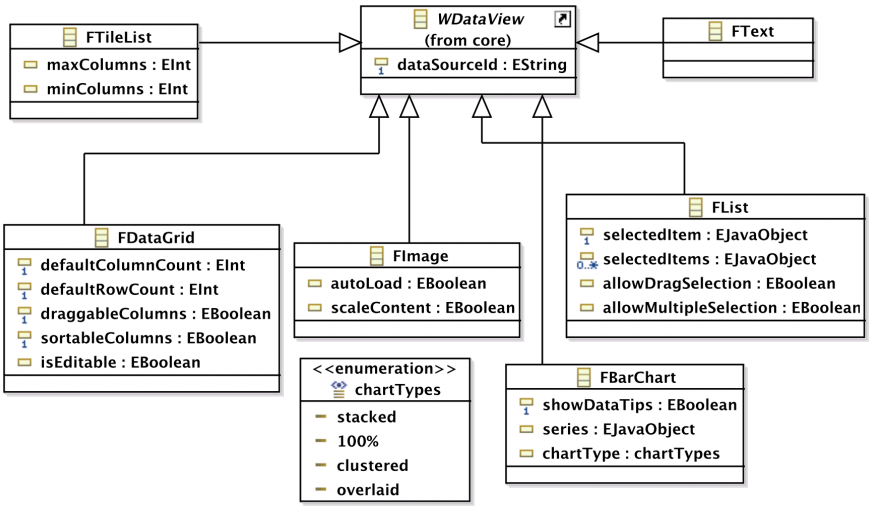


Fig. A.14: Entidades *DataView*

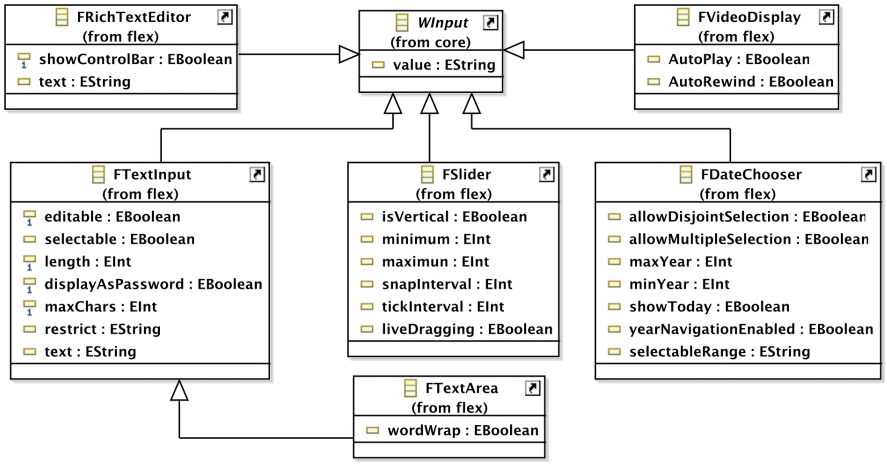


Fig. A.15: Entidades *Input* (1)

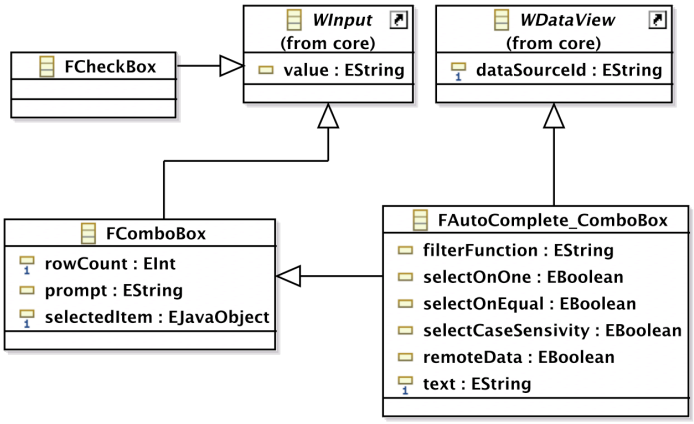


Fig. A.16: Entidades *Input* (2)

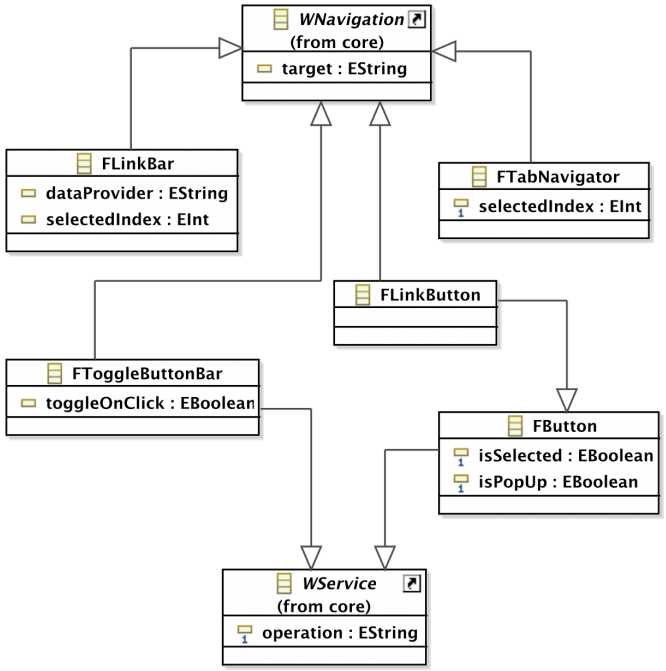


Fig. A.17: Entidades *Service* y *Navigation*

Además de los distintos *widgets,* el metamodelo también representa los eventos que proporciona la tecnología RIA. Estos eventos son necesarios para especificar las *Event rules* introducidas en la sección 5.2.2. Cada evento se define mediante una entidad con el mismo nombre encabezada por el prefijo "*EFon*". Los eventos definidos se agrupan en el paquete *flexEvents* del metamodelo. Todo evento se especializa a partir de la entidad *Event*, tal y como muestran la Fig. A.18 y la Fig. A.19. Cada *widget* se relaciona con los eventos que puede desencadenar. Por ejemplo, la entidad *FSlider* se relaciona con el evento *EFonThumbDrag* que se produce al desplazar la barra que representa el *widget*. Los eventos comunes a todos los *widgets*, como *EFonChange* y *EFonClick,* se relacionan con la entidad *FCommonWidget*. Al igual que en el caso anterior, ésta no es una representación completa de todos los eventos que proporciona *Adobe Flex,* puesto que solo se han considerado aquéllos utilizados en la presente tesis.

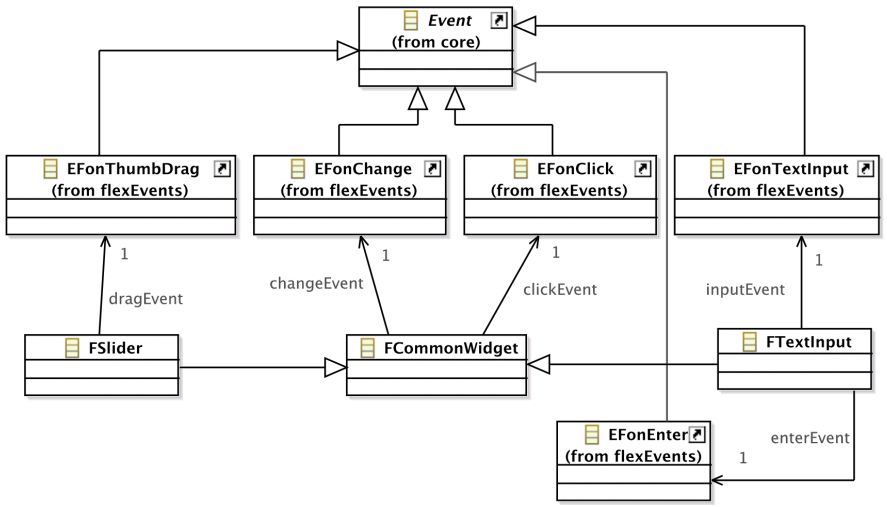


Fig. A.18: Eventos de *Adobe Flex* (1)

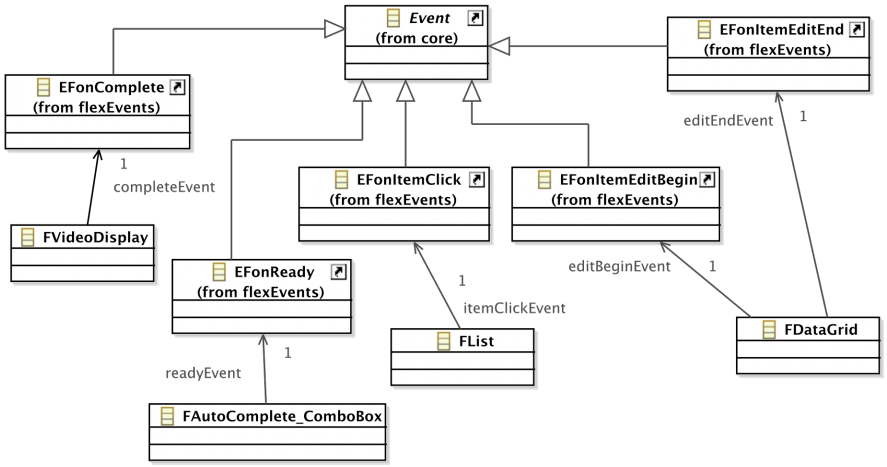


Fig. A.19: Eventos de *Adobe Flex* (2)

Sobre el autor

Francisco Valverde Giromé es un investigador del Centro de Investigación en Métodos de Producción de Software (ProS) perteneciente a la Universidad Politécnica de Valencia (UPV).

En julio del 2005 terminó sus estudios de Ingeniería Informática por la Universidad Politécnica de Valencia, incorporándose al grupo de investigación OO-Method mediante una beca de especialización. En abril del 2006, le fue concedida por el Ministerio de Educación una beca para la Formación del Profesorado Universitario (FPU) para la realización de sus estudios doctorales. Las líneas de investigación abordadas en su doctorado se enmarcan en las disciplianas de la Ingeniería Web y el desarrollo de software dirigido por modelos. Durante los años 2007-2009, impartió docencia de prácticas de la asignatura "Metodología y Tecnología de la Programación" en la Escuela Técnica Superior de Ingeniería Informática de la UPV.

Actualmente, desarrolla su investigación en la aplicación de los prinicipios de la Ingeniería del Software y del Modelado Conceptual al ámbito de la bioinformática. Para más información puede consultarse la página Web: http://fravalgi.webs.upv.es

Centro de Investigación en Métodos de Producción de Software

Centro de Investigación ProS
Universidad Politécnica de Valencia
Edificio 1F
Camino de Vera s/n
46022, Valencia, Spain
www.pros.upv.es

UNIVERSITAT POLITÈCNICA DE VALÈNCIA